Ingeniería Técnica de Telecomunicación: Telemática

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE GESTIÓN DE REPUTACIÓN PARA JXTA

Autor: Manuel Caballo Gil

Tutora: Florina Almenares Mendoza

Leganés, Junio de 2012

Título: Diseño e implementación de un sistema de gestión de reputación para JXTA

Autor: Manuel Caballo Gil

Directora: Florina Almenares Mendoza

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

En primer lugar me gustaría agradecer a mi familia la paciencia que ha tenido durante todos estos años, que les habrán parecido interminables, y también su apoyo, tanto en los buenos tiempos como en los momentos difíciles.

También quiero hacer mención a Celia, mi novia, a la que tengo que agradecer su inestimable ayuda, su comprensión, y haber estado a mi lado en todo momento. Del mismo modo le agradezco su impaciencia, que a menudo ha sido positiva, y fundamental para no rendirme y seguir trabajando.

A Florina, mi tutora, debo agradecerle el no haber tirado la toalla conmigo, su ayuda durante todo este proceso y la paciencia a lo largo de estos años de trabajo.

En esta serie de agradecimientos también me gustaría incluir a mis compañeros: Humberto, Rico, Jorge y tantos otros que me dejo en el tintero, por todos esos buenos momentos que hemos pasado en nuestro periplo por la universidad.

Por último, tengo que añadir una especial mención a mi madre, a la que debo en gran medida cómo soy, y a la que sin duda le hubiese gustado verme finalizando mis estudios universitarios.

Sinceramente, muchas gracias a todos.

RESUMEN

Las redes y aplicaciones *Peer-to-Peer* (*P2P)* se han extendido significativamente a lo largo de los últimos años, complementando a los sistemas centralizados tradicionales, y son un ejemplo actual en el que se dan continuas interacciones entre individuos desconocidos.

Por otro lado, la reputación es uno de los pilares fundamentales que llevan a las personas a confiar en otros individuos, ayudando a establecer medidas objetivas de la calidad del comportamiento de los mismos. Por lo tanto, la reputación es un parámetro fundamental que cada usuario debe ganarse de cara al resto de miembros de la red.

La mayoría de las aplicaciones *Peer-to-Peer* existentes hoy en día, como *eBay* o *BitTorrent*, implementan sistemas de gestión de reputación por motivos de seguridad, y para garantizar su correcto funcionamiento, por lo que serán una pieza clave dentro de las mismas. Los sistemas de reputación proporcionan información esencial para el cálculo de la confianza, como predicciones de comportamientos futuros basadas en acciones pasadas de un *peer*.

El objetivo de este proyecto consiste diseñar e implementar un sistema de gestión de reputación orientado a aplicaciones de anotación semántica dentro de comunidades, basado en medidas concretas de calidad que se traduzcan en un cálculo uniforme de la reputación de los individuos. Por tanto, este sistema se utiliza dentro de una red *Peer-to-Peer*, unificando los conceptos vistos anteriormente.

Para lograr este propósito, la implementación del sistema se ha sustentado en *JXTA*, un conjunto abierto de protocolos que modela el comportamiento de una red *Peer-to-Peer*. Además, se han tomado como referencia las directrices generales proporcionadas por el estudio teórico presentado en *Poblano*, un sistema de gestión de reputación y confianza diseñado específicamente para *JXTA*.

En este proyecto se aporta una interfaz genérica que puede ser utilizada por cualquier aplicación distribuida para beneficiarse de la funcionalidad proporcionada por un sistema de gestión de reputación descentralizado.

Palabras clave: Reputación, *Peer-to-Peer*, *JXTA*, *Poblano*

ABSTRACT

Peer-to-Peer (*P2P*) networks and applications have been increasingly extended during these last few years, complementing traditional centralized systems, and they are a current example in which interactions between unknown individuals continuously happen.

On the other hand, reputation is one of the cornerstones that lead people to deal with other individuals, helping to establish quality measures for those relationships. Therefore, reputation is a fundamental parameter that every user has to earn among other users.

Most part of the current *Peer-to-Peer* applications, such as *eBay* or *BitTorrent*, implement reputation systems for security reasons, and to ensure their proper operation, so they will be a key element for those applications. Reputation systems provide essential input for computational trust such as predictions of future behavior based on the past actions of a *peer*.

The main goal of this project is to design and implement a reputation management system aimed at semantic annotation systems in communities, based on specific quality measures which will translate into a uniform calculation of the reputation of the individuals. Therefore, this reputation management system is used on a *Peer-to-Peer* network, unifying both previous concepts.

To achieve this goal, the implementation is based on *JXTA*, a set of open protocols that models the behavior of a *Peer-to-Peer* network. Moreover, the general guidelines taken as a reference have been provided by the theoretical work shown in *Poblano*, a reputation and confidence management system specifically designed for *JXTA*.

This project offers a generic interface which can be used by any distributed application to profit from the features provided by a decentralized reputation management system.

Keywords: Reputation, *Peer-to-Peer*, *JXTA*, *Poblano*

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS

1.1. Motivación

La reputación está en el núcleo de todas las relaciones entre los seres humanos. Por este motivo, todo el mundo conoce lo que significa que alguien tenga buena o mala reputación. Los parámetros que miden la reputación vienen determinados por la opinión general que un determinado individuo inspire en la comunidad a la que pertenece, por lo que la descentralización es la naturaleza de la reputación, ya que cada individuo tiene su propia opinión, y la combinación de estas opiniones da lugar a una reputación.

En una red descentralizada, como por ejemplo las redes *Peer-to-Peer* (*P2P*), los usuarios pueden observar de donde reciben la información buscada. Del mismo modo éstos pueden comunicar la opinión tanto de la información que han adquirido, como de la fuente de la misma. Estas opiniones personales pueden ser almacenadas, intercambiadas y evaluadas. Se puede ir más allá, ya que esta información, una vez evaluada, puede ser utilizada como guía para buscar nueva información y nuevas fuentes que recomienden información. Este planteamiento generará una "Web de confianza" descentralizada y personalizada.

Cuando este modelo de reputación descentralizado es implementado en una topología de red *Peer-to-Peer*, la confianza entre *peers* y su reputación empiezan a reflejar fielmente las relaciones del mundo real, con las cuales los usuarios están familiarizados, y permitiendo a los desarrolladores de *software* trabajar sobre interfaces inteligibles y utilizables de estos modelos de reputación. De esta forma, la confianza se convierte en un contrato con implicaciones sociales entre los participantes. Cada uno de estos participantes deberá tratar de ganarse una buena reputación de cara a sus semejantes, lo que constituye las bases de las relaciones personales.

En la actualidad, estas implementaciones pueden encontrarse en multitud de aplicaciones, como mercados electrónicos y comunidades de Internet, convirtiéndose en un componente de gran importancia para el estímulo del buen comportamiento entre los participantes, y para la sanción de las malas prácticas llevadas a cabo por los nodos.

Este punto de partida hace interesante el estudio de este tipo de sistemas de gestión de reputación, y nos lleva a plantear la posibilidad de implementar un sistema propio, basado en alguno de los planteamientos teóricos existentes en este campo, y orientado a aplicaciones de anotación semántica dentro de comunidades *Peer-to-Peer*. En concreto, este proyecto pretende dotar al proyecto *ITACA* [1] de un modelo de gestión de la reputación para las anotaciones semánticas en la *Web*.

La *Web* semántica necesita que los contenidos tengan anotaciones semánticas que los expliquen formalmente. La propuesta del proyecto *ITACA* es hacer un sistema que aproveche las consultas en buscadores de la *Web* para hacer anotaciones semánticas y almacenar las anotaciones de forma distribuida utilizando tecnologías *P2P*.

Como en cualquier otro enfoque manual, las anotaciones realizadas por usuarios, no necesariamente expertos, pueden ser incorrectas o subjetivas. Por este motivo, es necesario introducir un modelo de confianza que permita a usuarios, y aplicaciones software que operen en su nombre, determinar el nivel de confianza otorgable a una determinada anotación. Con esta base se quiere dar soporte a la formación de comunidades semánticas en redes *Peer-to-Peer*, prestando especial atención a los sistemas de reputación como medio para facilitar el acceso y la formación de confianza situacional.

Esta memoria tratará de esta implementación, en la que se podrán ver con detalle todos los aspectos de la misma, desde el planteamiento del diseño hasta el desarrollo llevado a cabo.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es implementar un sistema descentralizado de gestión de reputación sobre una red *Peer-to-Peer*, en concreto *JXTA*. Este modelo representará no sólo las relaciones de confianza entre *peers* y su reputación en una comunidad, sino también la confianza entre *peers* y contenidos. En este proyecto se propondrán los protocolos encargados de distribuir dicha información de reputación, así como los algoritmos utilizados para su actualización.

Para ello, un primer objetivo específico consiste en estudiar la plataforma *JXTA* en dos vertientes: la primera de ellas será el modelo de red *Peer-to-Peer* que implementa, y en segundo lugar los métodos que proporciona para la comunicación entre nodos, ambos con el propósito de integrarlo con el sistema de gestión de reputación.

Otro objetivo específico es dotar al sistema de los interfaces necesarios para que cualquier aplicación externa sea capaz de manejar la funcionalidad de toda la gestión de los valores de reputación de *peers* y contenidos. De esta forma cualquier aplicación, no necesariamente un usuario, podrá actuar como un nodo de la red e interactuar con el resto de *peers*.

El tercer objetivo específico consiste en diseñar e implementar una aplicación de prueba que sea capaz de permitir a un usuario interaccionar con el sistema de gestión de reputación desarrollado, con intención de mostrar su funcionamiento y hacerle capaz de actuar como un nodo más de la red.

Por último, y como objetivo específico final, se plantea realizar las pruebas necesarias para verificar el correcto funcionamiento del sistema de gestión de la reputación. Esto proporcionará la base de la evaluación del mismo, y servirá para comprobar el cumplimiento de los requisitos técnicos.

Se puede considerar un objetivo complementario el confeccionar una documentación clara que respalde al desarrollo del proyecto, y que permita la comprensión del mismo. Esto facilitará el estudio teórico del proyecto y servirá de referencia para resolver cualquier duda sobre el sistema implementado.

1.3. Estructura de la memoria

A continuación se presentará una relación de todos los capítulos en los que se divide el presente proyecto, como presentación de su estructura y para describir el contenido de cada uno de ellos:

- **Capítulo 2: Estado del arte**

Este capítulo trata sobre las tecnologías utilizadas para la implementación del proyecto, las diferentes alternativas que se han presentado para su realización y una introducción detallada a cada una de ellas.

- **Capítulo 3: Descripción general del sistema**

En este capítulo se hace una presentación de la solución final adoptada para la implementación del proyecto, explicando los conceptos básicos que determinan su funcionamiento y describiendo las estructuras necesarias para su desarrollo.

- **Capítulo 4: Implementación del sistema**

Explica la implementación final del proyecto, mostrando la división del sistema en clases e interfaces, y explicando su relación con los módulos del sistema. También se detallan los algoritmos básicos que serán la base de su funcionamiento.

- **Capítulo 5: Pruebas del sistema**

En este quinto capítulo se presentan todas las pruebas a las que se ha sometido cada módulo del sistema, así como las referentes al sistema completo, sin olvidar la aplicación de prueba. En este capítulo se mostrarán los resultados obtenidos en dicha tarea.

- **Capítulo 6: Historia del Proyecto**

En este capítulo se detalla la evolución del proyecto desde su planteamiento hasta su conclusión, mostrando todas las fases por las que ha ido pasando y presentando los cambios relevantes que ha sufrido en el proceso.

- **Capítulo 7: Conclusiones y trabajos futuros**

Este capítulo sirve de evaluación del cumplimiento de los objetivos que se marcaron al inicio del proyecto, presentando los resultados obtenidos y definiendo nuevas líneas de investigación sobre las que se puede continuar el trabajo.

- **Anexo A: Presupuesto**

Este anexo sirve para abordar el aspecto económico del proyecto, desglosando todos los costes económicos y temporales del mismo. Los datos económicos servirán para dar una idea del precio del proyecto en el mercado, mientras que los temporales ayudarán a hacerse una idea de la planificación.

- **Anexo B: Planificación y Diagrama de Gantt**

Este anexo muestra los diagramas correspondientes a la estructuración temporal del proyecto, especificando las distintas tareas que han sido necesarias para la consecución del mismo y colocándolas en una línea temporal que las relacione a todas ellas.

- **Anexo C: Manual de la aplicación de prueba**

Este anexo trata de explicar la estructura y el funcionamiento de la aplicación de prueba incluida en el proyecto para la utilización del sistema, de forma que cualquier usuario pueda manejarla sin problemas.

- **Anexo D: Manual del desarrollador**

El último anexo trata de ser una referencia para que cualquier desarrollador pueda conocer de forma pormenorizada los métodos y atributos empleados en la implementación del sistema, facilitando su trabajo.

CAPÍTULO 2: ESTADO DEL ARTE

2.1. Peer-to-Peer

2.1.1. Introducción

La arquitectura *Peer-to-Peer*, o de forma abreviada *P2P*, es un tipo de red en la que cada estación de trabajo tiene las mismas responsabilidades y capacidades que el resto. Este modelo es completamente diferente a la arquitectura cliente-servidor, en la que algunos equipos se dedican exclusivamente a servir a otros. Así, la arquitectura *P2P* es un sistema distribuido en el que no existe un control centralizado ni ninguna estructura jerárquica.

Los sistemas *Peer-to-Peer* han sido descritos ampliamente en multitud de documentos. A continuación, se presentarán dos de las definiciones que muestran los conceptos de compartición de recursos, autoorganización, descentralización e interconexión, básicos para este tipo de sistemas:

"Una arquitectura de red distribuida puede ser llamada red Peer-to-Peer si los participantes comparten parte de sus propios recursos hardware (procesador, almacenamiento, capacidad de enlace con otras redes, impresoras, etc.) Estos recursos compartidos son necesarios para proporcionar los servicios y los contenidos que la red ofrece (p. e. compartición de ficheros). Todos ellos son accesibles por otros peers".[2]

"Los sistemas Peer-to-Peer son sistemas distribuidos consistentes en nodos interconectados capaces de organizarse en diferentes topologías de red con el objetivo de compartir recursos, como contenidos, ciclos de CPU, almacenamiento y ancho de banda, y que son capaces de adaptarse frente a fallos y permitiendo un número variable de nodos mientras se mantienen unos niveles aceptables de rendimiento e interconectividad, sin necesidad de intermediarios ni servicios o autoridades centralizadas".[3]

Tal y como se verá en los siguientes apartados, el modelo representado por la arquitectura *Peer-to-Peer* forma una buena base para la implementación del sistema de gestión de reputación sobre el que trata este proyecto. Este sistema se basa en una arquitectura muy similar a la descrita por las redes *P2P*, en la que los *peers* compartirán información de reputación de forma totalmente independiente.

Cada uno de los nodos se encargará de crear, actualizar, mantener y compartir información de reputación de contenidos y de *peers*, para construir de esta forma un sistema completamente descentralizado. Estos *peers* serán capaces de actuar sin la necesidad de ningún tipo de órgano gestor común, actuando como iguales y suponiendo sólo una parte de la totalidad del sistema. Esto implica que la ausencia de alguno de los usuarios en la red no supone ningún cambio en el funcionamiento ni en el rendimiento del mismo.

De esta forma, se conseguirá hacer uso de una tecnología en alza, como son las redes *Peer-to-Peer*, que hará de este sistema de gestión de reputación una aplicación basada en las nuevas tecnologías, evitando caer en los viejos modelos cliente-servidor. Esto facilitará la creación de un sistema de gestión de reputación totalmente distribuido.

2.1.2. Propiedades

Las propiedades de los sistemas *P2P* pueden ser clasificadas atendiendo a diversos criterios, aunque se puede confeccionar un listado de las más comunes que se pueden encontrar:

- **Compartición de recursos:** Cada *peer* debe proporcionar recursos para el correcto funcionamiento del sistema *Peer-to-Peer*. Idealmente, esta compartición debe ser proporcional al uso que haga el *peer* del sistema, aunque siempre puede aparecer el problema del denominado *free-rider*.
- **Conectividad:** Todos los nodos deben estar interconectados con otros *peers* en el sistema *P2P*.
- **Descentralización:** El comportamiento de un sistema *P2P* está determinado por la acción de los nodos que lo componen, y no existe ningún punto de control central. A pesar de esto, hay sistemas que refuerzan su seguridad utilizando sistemas de autenticación centralizados.
- **Simetría:** Los nodos asumen papeles iguales dentro del sistema *P2P*. En algunos sistemas esta propiedad no es demasiado estricta, como el caso de *JXTA*, que se verá más adelante, permitiendo la existencia de papeles especiales como el de *superpeer*.
- **Autonomía:** La participación del *peer* dentro del sistema *P2P* está determinado localmente, y por lo tanto no hay ningún elemento administrativo común para el sistema completo.
- **Autoorganización:** La organización de los sistemas *P2P* se incrementa según avanza el tiempo debido a los nuevos conocimientos y operaciones de cada *peer*.
- **Escalabilidad:** Este es un requisito a priori para cualquier sistema *P2P* en los que potencialmente pueden existir millones de nodos conectados simultáneamente. Implica que los recursos utilizados de cada nodo, así como el tiempo de repuesta a peticiones no debe aumentar linealmente con el tamaño de la red, sino que debe ser menor.
- **Estabilidad:** Aunque el sistema presente el máximo grado de variabilidad, este debe ser estable, por ejemplo debe ser capaz de encaminar cualquier mensaje de forma determinista dentro de un número determinado de nodos intermedios.

2.1.3. Clasificación

Los diferentes diseños de redes *P2P* han generado varios métodos de clasificación. Por ejemplo, los sistemas de compartición de ficheros se han dividido en generaciones. La primera de ellas está formada por sistemas híbridos que combinan servidores con encaminamiento *P2P*, mientras que la segunda generación la componen arquitecturas completamente descentralizadas. Los sistemas *P2P* anónimos están considerados como la tercera generación. Algún ejemplo de esta generación son *Freenet* [4] o *I2P* [5].

La clasificación de los sistemas *P2P* tiene varios inconvenientes, ya que deja de lado características importantes de los mismos por las cuales podrían ser clasificados, y no explica qué requisitos deberían cumplir los sistemas de generaciones futuras. En cualquier caso, sistemas pertenecientes a las tres generaciones han estado en funcionamiento simultáneamente.

Otro método de clasificación para las redes *P2P* es el que las divide en estructuradas y no estructuradas. Las redes no estructuradas pueden ser subclasificadas según el modo de propagación de las peticiones, distribución de los nodos dentro de la comunidad de *peers* y diferentes formas de establecer relaciones entre *peers*. Por su parte, las redes estructuradas pueden ser clasificadas de acuerdo con múltiples características como:

- Número máximo de saltos para encaminar una petición (multisalto, salto único, salto variable).
- Algoritmo de encaminamiento (por prefijo, *XOR*, distancia geométrica, diferencia en el espacio de direcciones, distancia semántica).
- Nivel del nodo dentro de la red (nivel constante, nivel logarítmico).
- Geometría de la red.
- Tipo de búsqueda (Iterativa o recursiva, en serie o en paralelo).

Más allá de las categorías principales, es decir, las estructuradas y no estructuradas, se pueden encontrar otras categorías como las redes jerárquicas, redes federadas, redes de servicio (específicas para desplegar servicios de red), redes semánticas (que encaminan peticiones atendiendo a relaciones semánticas) y redes que proporcionan servicios para nodos móviles. Para visualizar mejor esta división en categorías se puede consultar la figura 1:

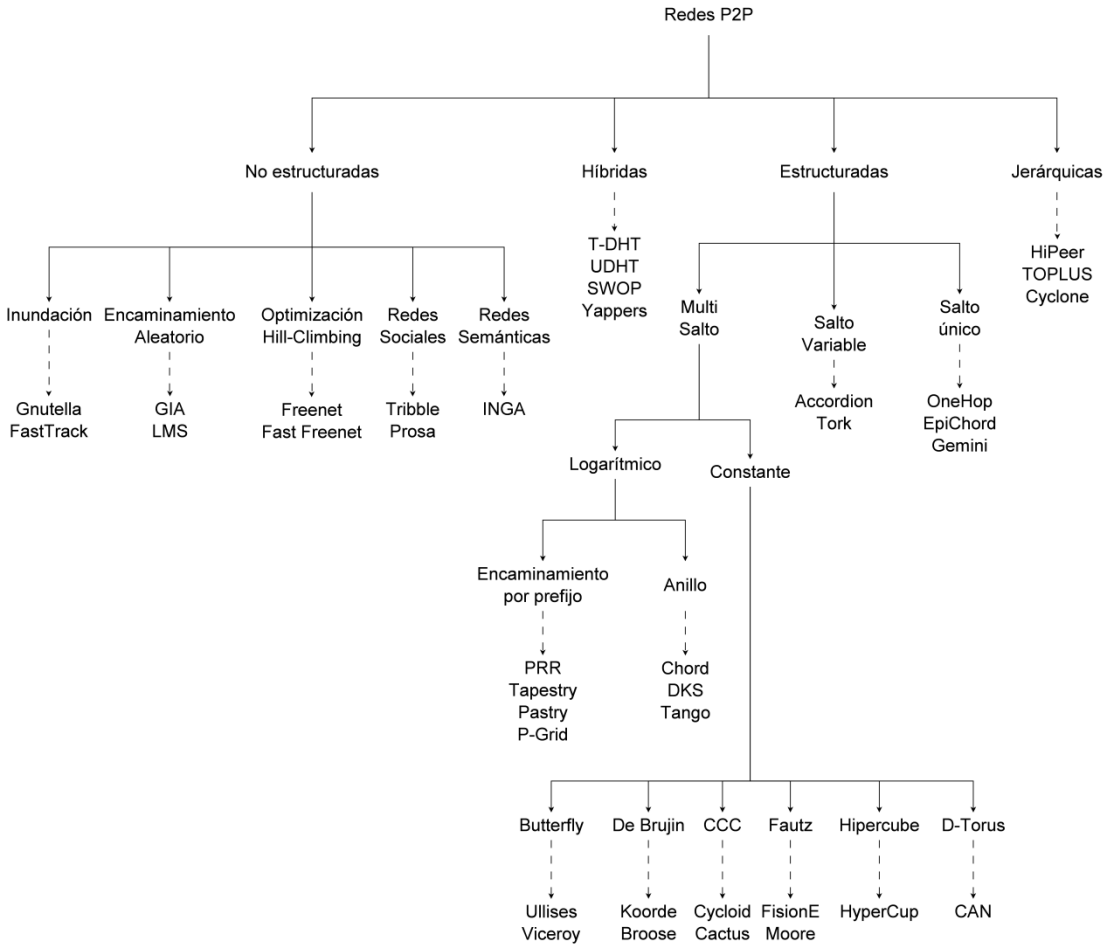


Figura 1: Clasificación de Redes P2P

Una red no estructurada es *"una red en la que un nodo depende únicamente de sus nodos adyacentes para entregar mensajes a otros nodos en la red"*[6]. Ejemplos de estrategias de propagación de mensajes en este tipo de redes son la inundación (*flooding*) y el encaminamiento aleatorio (*random walk*). Importantes investigaciones en el campo de las redes no estructuradas tratan de mejorar el diseño de búsquedas eficientes, centrándose en la propagación y procesado de peticiones.

Otra vía de investigación dentro del campo de las redes no estructuradas es el estudio del gráfico óptimo de la red, es decir, la figura que forman los nodos dentro de la red, así como los algoritmos descentralizados que permiten formar y mantener estos gráficos en el entorno de una población de *peers* continuamente cambiante.

Por otra parte, una red estructurada es *"una red en la cual los nodos mantienen cooperativamente la información de encaminamiento para poder alcanzar a todos los nodos de la red"*[6]. Comparados con las redes no estructuradas, las estructuradas proporcionan un número limitado de mensajes necesarios para encontrar cualquier contenido en la red.

Esto es especialmente importante para los casos de búsquedas dirigidas a contenidos poco populares. Para proporcionar un encaminamiento determinista, los *peers* están situados en un espacio virtual de direcciones, organizándose según una geometría específica y definiendo un algoritmo de encaminamiento para el reenvío de mensajes.

Cada *peer* tiene una tabla local de encaminamiento que será utilizada por el algoritmo de encaminamiento correspondiente. Esta tabla local se inicializa cuando el nodo se une a la red mediante un proceso de *bootstrapping*. Los *peers* intercambian periódicamente sus tablas de rutas como parte del mantenimiento de la red.

La mayoría de las redes *P2P* estructuradas utilizan sistemas de encaminamiento basado en claves en las cuales *"un conjunto de claves está asociado con direcciones pertenecientes a un espacio de direcciones, de forma que el peer más próximo a una dirección dada será el encargado de almacenar las claves correspondientes, y el algoritmo de encaminamiento tratará dichas claves como direcciones"*[6]. Una *hashtable* distribuida (*DHT*) es un tipo de red estructurada que utiliza el encaminamiento basado en claves para almacenar y obtener índices, y en la cual cada *peer* tiene por cometido el mantenimiento de parte de los índices de la *DHT*.

Debido a que el espacio de direcciones es virtual y las direcciones asignadas a los *peers* son aleatorias, puede ser que *peers* que estén muy próximos físicamente en la red puedan estar muy separados en la estructura formada. Esto proporciona al sistema un alto grado de tolerancia a fallos, pero por contra provoca una pérdida en el rendimiento. Por este motivo existen redes que sí se fijan en la topología física de la red, y que utilizan medidas de proximidad entre *peers* para que sean también vecinos en la red estructurada subyacente.

2.1.4. Aplicaciones y usos

En este apartado se tratará de mostrar la importancia de las redes *P2P* a través de algunas aplicaciones de ejemplo más o menos conocidas. Esto permitirá formar una idea de las aplicaciones prácticas de este tipo de redes:

- ***Chord*** [7]

Es un popular algoritmo *P2P* estructurado creado por varios desarrolladores del Instituto Tecnológico de Massachusetts [8] que se basa en la implementación de una *DHT* (*Distributed Hash Table*), permitiendo que cada nodo mapee un conjunto de claves correspondientes a contenidos para facilitar su búsqueda.

- ***Content Addressable Network* (*CAN*)** [9]

Es una infraestructura *P2P* distribuida, descentralizada y estructurada que podría considerarse muy similar a *Chord*.

CAN utiliza un espacio de coordenadas virtual cartesiano d-dimensional para alojar tanto las claves correspondientes a los contenidos como los identificadores de los nodos. El mapeo claves/nodos en el espacio d-dimensional se realiza a través de una función de *hash*, con la particularidad de que es necesario un protocolo adicional de mantenimiento para remapear periódicamente las claves. *CAN* optimiza el reenvío de peticiones calculando la ruta en base al mejor *RTT* (*Round Trip Time*) de cada uno de sus vecinos, hecho transparente para el propio peticionario. Esto implica que el proceso de reenvío no es verificable por el peticionario, lo que hace vulnerable al sistema frente a los ataques contra el algoritmo de encaminamiento.

- ***Kademlia*** [10]

Es la implementación de una *hashtable* distribuida para una red *P2P* descentralizada y no estructurada que ha sido utilizada para el intercambio de ficheros. El algoritmo utilizado por *Kademlia* se basa en el cálculo de la distancia entre los identificadores de dos nodos. Esta distancia se emplea para llenar una lista de identificadores con las peticiones para las que el nodo recibe respuesta. Es un sistema resistente a ciertos tipos de ataque de denegación de servicio, ya que de la lista no pueden borrarse los nodos que siguen siendo válidos. *Kademlia* puede realizar múltiples búsquedas paralelas para la misma petición. Por ejemplo, está implementado en la red descentralizada *Kad* [11].

- ***Napster*** [12]

Fue uno de los primeros experimentos en el ámbito de las redes *P2P*, junto con otros como *Gnutella* o *Freenet*. Es un sistema híbrido ya que solo la capacidad de descarga es distribuida, mientras que la búsqueda de contenidos se hace a través del servidor central de índices. *Napster* fue cerrado debido a la violación de derechos de contenidos digitales, sin embargo ha sido un gran paso adelante para el desarrollo de programas descentralizados de compartición de ficheros como *KaZaA* [13].

- ***BitTorrent*** [14]

El protocolo *BitTorrent* es actualmente una de las plataformas de distribución de contenido digital líder, basándose en la filosofía de las redes *P2P*. Funciona con una topología de red no estructurada, existiendo clientes basados en *DHT* como por ejemplo *uTorrent* [15]. *BitTorrent* no se puede considerar como un sistema absolutamente descentralizado, ya que necesita la utilización de sitios web para publicar los contenidos que la gente está compartiendo. A través de los enlaces de dichas páginas *web*, el usuario puede acceder a una lista de *peers* que comparten dicho fichero y empezar a descargarlo de varios de ellos simultáneamente. Esta capacidad posibilita que las descargas se hagan rápidamente y por ello es una de las razones del éxito del sistema. Además del clásico uso de *BitTorrent* para compartir ficheros, un número cada vez mayor de organizaciones está utilizando el sistema para la distribución totalmente legal de contenidos como, por ejemplo, programas de televisión.

- ***Gnutella*** [16]

Es un sistema desarrollado un año después que *Napster* aunque, a diferencia de este, es completamente distribuido. Ganó una importante popularidad tras el cierre de *Napster* en 2001. En sus inicios tenía un grado de escalabilidad que dejaba bastante que desear, por lo que las mejoras introducidas en *Gnutella2* [17] como por ejemplo la introducción de supernodos y más tarde de concentradores (*hubs*) centrales, permitió resolver el problema. Aunque en la actualidad es una red que sigue en funcionamiento, la propagación de otro tipo de redes hace que *Gnutella* tenga menos de un 10% del total del tráfico de red *P2P*.

- ***JXTA*** [18]

Es una plataforma libre *P2P* definida como un conjunto de protocolos basados en *XML*. Es una robusta implementación de una red *P2P* diseñada para permitir las comunicaciones descentralizadas entre una amplia variedad de dispositivos. *JXTA* es una plataforma modular, que consta de varios bloques constituyentes independientes que permiten el desarrollo de un gran número de servicios y aplicaciones, definiendo simplemente un conjunto de protocolos que deben seguir. Está optimizado para la variación continua de los *peers* presentes en la red, soportando grandes cambios en poco tiempo, a diferencia de otros sistemas como *Chord* o *CAN* en los que se busca la optimización del tiempo de búsqueda. Por las características y el funcionamiento que ofrece, la red *P2P JXTA* será la elegida para la implementación del sistema de gestión de reputación que se desea implementar.

2.2. JXTA

2.2.1. Introducción

JXTA [18] es un conjunto de protocolos *Peer-to-Peer* (*P2P*) abiertos y generales que permiten a cualquier dispositivo conectado en red –sensores, teléfonos móviles, PDA, ordenadores portátiles, estaciones de trabajo, servidores y supercomputadores- comunicarse y colaborar mutuamente como *peers*. Los protocolos definidos por *JXTA* son independientes del lenguaje de programación, y existen múltiples implementaciones para diferentes entornos. La utilización de *JXTA* por todos ellos significa que pueden interactuar completamente. Las principales implementaciones de *JXTA* son las siguientes:

- **JXSE:** Implementa *JXTA* para *Java Platform, Standard Edition* (*Java SE*).
- **JXME:** Implementa *JXTA* para *Java Platform, Micro Edition* (*Java ME*).
- **JXTA-C:** Implementa *JXTA* para *C*, *C++* y *C#*

JXTA es una plataforma con años de experiencia, propuesta por la desaparecida *Sun Microsystems* en el año 2001 y cuya especificación fue remitida al *IETF* [19] (*Internet Engineering Task Force*) en Junio de 2002. Este organismo declinó asignar a *JXTA* un grupo de trabajo específico, pero se remitió al *IRTF* (*Internet Research Task*

Force) donde se incluyó en el grupo de trabajo *Peer-to-Peer* de este organismo [20]. Además de esto, un grupo de desarrolladores independiente [21] ha seguido con la implementación de nuevas versiones de esta plataforma, siendo la versión más reciente la 2.7, que en el momento de la escritura de esta memoria sigue en pruebas.

Dado el crecimiento imparable de la *Web*, tanto en contenidos como en dispositivos conectados, el uso de aplicaciones *Peer-to-Peer* se ha extendido exponencialmente. Algunos ejemplos de sobra conocidos incluyen aplicaciones de compartición de ficheros, computación distribuida, y otros que proporcionan servicios de mensajería instantánea. Aunque todas estas aplicaciones tienen cometidos y tareas distintos, todos ellos comparten bastantes propiedades, como el descubrimiento de *peers*, búsquedas, y transferencia de datos o ficheros.

Actualmente, el desarrollo de aplicaciones *P2P* no es eficiente, ya que los desarrolladores tienen que duplicar implementaciones para resolver los mismos problemas en diferentes aplicaciones. De hecho, la mayoría de las aplicaciones *Peer-to-Peer* son específicas para una única plataforma, siendo incapaces de comunicarse y compartir datos con otras aplicaciones.

El diseño fundamental de *JXTA* es proveer una plataforma que englobe todas las funcionalidades básicas de las redes *P2P*. De esta forma, *JXTA* supera los defectos potenciales de muchos de los sistemas *P2P* actuales, ofreciendo soluciones para algunos aspectos como:

- **Interoperabilidad:** La tecnología *JXTA* está diseñada para que *peers* proveedores de servicios *P2P* se localicen y comuniquen entre sí independientemente del direccionamiento de la red y de los protocolos físicos.
- **Independencia de la plataforma:** *JXTA* está diseñado para ser independiente de los lenguajes de programación, protocolos de nivel de transporte utilizados por la red, y plataformas de despliegue.
- **Ubicuidad:** *JXTA* está diseñado para poder ser accesible para cualquier dispositivo digital, no solamente ordenadores personales o dispositivos específicos.

Una característica común a todos los *peers* de una red *P2P* es que a menudo forman parte de redes comunes, y por lo tanto los dispositivos conectados no disponen de direcciones estáticas, por ejemplo utilizando *DHCP*; por este motivo, ya que los *peers* no están continuamente conectados y no siempre cuentan con la misma dirección de red, estos están fuera del alcance del *DNS*. Sin embargo, *JXTA* otorga a los *peers* un esquema de direccionamiento para cada *peer* único y global que es independiente de los servicios de nombres tradicionales. Mediante el uso de identificadores *JXTA*, un *peer* puede cambiar entre redes físicas, protocolos de transporte y direcciones de red, incluso estar temporalmente desconectados de la red, y aún así poder ser localizado por otros *peers*.

Así, se puede decir que *JXTA* es una plataforma abierta para procesamiento *Peer-to-Peer* (*P2P*) proporcionando los bloques básicos y los servicios necesarios para permitir la interconexión entre cualquier aplicación en cualquier sitio.

El nombre de *JXTA* no es un acrónimo, sino que es una forma abreviada del término yuxtaponer (en inglés *juxtapose*), lo que deja clara la idea de que el procesamiento *P2P* está en yuxtaposición con las arquitecturas cliente-servidor o basadas en la *Web*, que son los modelos tradicionales de arquitectura distribuida.

JXTA proporciona un conjunto común de protocolos abiertos respaldados con implementaciones abiertas de referencia para poder desarrollar aplicaciones *Peer-to-Peer*. De esta forma *JXTA* permite normalizar la forma en la que los *peers*:

- Descubren otros *peers*.
- Se organizan en grupos.
- Anuncian y descubren recursos en la red.
- Se comunican con otros *peers*.
- Monitorizan otros *peers*.

Como se ha podido ver en la presentación de esta tecnología, *JXTA* representa una plataforma idónea para la implementación del sistema de gestión de reputación objeto de este proyecto. Además de contar con un soporte que hará a este sistema independiente del dispositivo, red o sistema operativo en el que se despliegue, ofrece una total integración con las redes de tipo *P2P* que se vieron en la sección anterior, y que tomamos como referencia para el mismo.

De esta forma se ha conseguido sentar las bases sobre las que descansará este sistema de gestión de la reputación, en lo que a redes y a intercomunicación de usuarios se refiere. *JXTA* dotará al sistema de un punto de acceso al mundo de las redes *Peer-to-Peer* a través de un sencillo conjunto de protocolos que se irán describiendo más adelante, y ofrecerán una serie de conceptos interesantes para su implementación que se describirán en el apartado siguiente.

2.2.2. Conceptos importantes

- ***Peer***

Un *peer* es cualquier entidad conectada a la red que implemente uno o más de los protocolos *JXTA*. Los *peers* pueden estar en sensores, teléfonos móviles, PDA, así como ordenadores personales, servidores y supercomputadoras. Cada *peer* actúa independientemente y de forma asíncrona con respecto al resto de *peers* y está identificado unívocamente por un *PeerID*.

Los *peers* publican una o más direcciones de red para que puedan ser utilizadas por los protocolos *JXTA*. Cada dirección publicada es anunciada como un *endpoint* del *peer*, que identifica dicha dirección. Los *endpoints* son utilizados por los *peers* para establecer comunicaciones punto a punto entre dos nodos.

Las comunicaciones directas punto a punto no son siempre posibles, por lo que *peers* intermedios pueden ser utilizados para encaminar mensajes a otros *peers* que están separados debido a las divisiones físicas entre redes. Estas fronteras en las redes pueden ser límites naturales, como los existentes entre redes *Ethernet* y *Bluetooth*, o creados por configuraciones especiales de red. Estas barreras artificiales incluyen *NAT*, *firewalls* y *proxies*. La utilización de estos *peers* intermedios puede cambiar a lo largo del tiempo sin tener ningún impacto en la aplicación *JXTA*.

Los *peers* pueden ser de tres tipos:

- *Minimal-Edge peers:* Son aquellos que implementan únicamente los servicios del núcleo de *JXTA* y pueden utilizar a otros *peers* como *proxies* para interactuar de forma completa en la red *JXTA*. Ejemplos típicos de estos *peers* pueden ser sensores y dispositivos domóticos.
- *Full-Edge peers:* Son aquellos que implementan los servicios del núcleo de *JXTA* y que pueden participar en la red utilizando cualquiera de los protocolos estándar de *JXTA*. Estos son la mayoría de los *peers* presentes en la red, y pueden ser teléfonos, ordenadores personales, servidores, etc.
- *Superpeers:* Son aquellos que implementan y aprovisionan a otros *peers* de recursos para soportar el despliegue completo de la red *JXTA*. Existen tres funciones clave que corresponden a *superpeers*, aunque un *peer* puede implementar una o más de estas funciones:
  - ***Relay*:** Se utilizan para almacenar y redirigir mensajes entre *peers* que no tienen una conexión directa debido a la presencia de cortafuegos o *NAT*.
  - ***Rendezvous*:** Mantienen un índice global de los anuncios de la red y facilita las búsquedas de estos anuncios al resto de *peers*. También interviene en la propagación de mensajes.
  - ***Proxy*:** Son utilizados por los *minimal-Edge peers* para tener acceso a las funcionalidades completas de *JXTA*.

- ***Peer Group***

Un *Peer Group* o grupo de *peers*, es un conjunto de nodos que tienen intereses en común o comparten varios servicios. Los *peers* se autoorganizan en grupos, cada uno de los cuales se identifica dentro de la red *JXTA* por un *PeerGroupID* único. Cada grupo establece las políticas para poder ser miembro, que incluyen desde las más abiertas (cualquiera puede unirse) hasta las más seguras (pidiendo credenciales).

Los *peers* pueden pertenecer a varios grupos simultáneamente. Por defecto, el primer grupo que se instancia, y al que todos los *peers* pertenecen, es el llamado *Network Peer Group*. Una vez que los *peers* pertenecen a este grupo pueden elegir unirse a más grupos en cualquier momento.

Los protocolos *JXTA* dictan cómo los *peers* deben publicar, descubrir, unirse y monitorizar grupos, aunque no especifican cuando ni por qué son creados. El hecho de unirse a un grupo significa instanciar todos los servicios definidos por el mismo. Hay varias motivaciones por las que se crean los grupos:

- Para crear un entorno seguro.
- Para crear un entorno dirigido a un ámbito concreto.
- Para crear un entorno de monitorización.

- ***Message***

Los servicios y aplicaciones *JXTA* se comunican utilizando mensajes. Un mensaje *JXTA* es la unidad básica de intercambio de información entre *peers*. Cada protocolo *JXTA* se define como un conjunto de mensajes que los *peers* se intercambian.

El uso de mensajes *XML* para definir los protocolos *JXTA* permite a diferentes tipos de *peers* utilizar los mismos protocolos. Ya que la información del mensaje está etiquetada en formato nombre-valor, cada *peer* puede implementar el protocolo a su manera, siempre que respete el formato.

Los protocolos *JXTA* definen dos tipos de representaciones para los mensajes en la red: *XML* y binarios. Estas representaciones son los formatos de datos utilizados para transmitir la información entre *peers*. Cada uno de estos formatos es utilizado para aprovechar todas las posibilidades del nivel de transporte subyacente. Por ejemplo, la implementación *JXSE* utiliza en formato binario del mensaje.

- ***Advertisement***

Los *advertisements*, o anuncios, son documentos de metadatos utilizados por los protocolos *JXTA* para describir recursos. Los anuncios se utilizan para describir *peers*, grupos, contenidos, etc. Los anuncios *JXTA* son representados en formato *XML*.

```
<?xml version="1.0" ?>
<!DOCTYPE jxta:PipeAdvertisement>
- <jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
    <Id>urn:jxta:uuid-
      596162616461626 14E504720503250338E3E786229EA460DADC1A176B69B731504</Id>
    <Type>JxtaUnicast</Type>
    <Name>TestPipe</Name>
  </jxta:PipeAdvertisement>
```

Figura 2: Ejemplo de Anuncio JXTA

Cada anuncio es publicado en la red con un determinado tiempo de vida que indica la disponibilidad de su recurso asociado. La utilización de estos tiempos de vida permite el borrado de recursos obsoletos sin la necesidad de ningún control centralizado. En cualquier caso, un anuncio puede ser publicado de nuevo (antes de que expire el tiempo del antiguo) para extender así el tiempo de vida del recurso.

- ***ID***

Los protocolos *JXTA* a menudo necesitan hacer referencia a *peers*, grupos, *pipes* y otros recursos *JXTA*. Estas referencias son representadas en los protocolos como *JXTA ID* o identificadores. Estos identificadores proporcionan referencias unívocas a las diferentes entidades *JXTA*, de forma que existen seis tipos en función de para qué son utilizados: grupos, *peers*, *pipes*, contenidos, clases de módulo y especificaciones de módulo. Cualquier identificador *JXTA*, independientemente del tipo que sea, debe cumplir los siguientes requisitos:

- **No ser ambiguos:** Deben ser referencias completas al recurso que describen.
- **Ser únicos:** Cada identificador debe hacer referencia a un único recurso.
- **Respetar los cánones:** Referencias a los mismos recursos deberán ser codificadas con los mismos identificadores.
- **Ser opacos:** El contexto en el que se utiliza un identificador deberá ser suficiente para determinar su tipo.

2.2.3. *Arquitectura JXTA*

La arquitectura software de JXTA está dividida en tres capas:

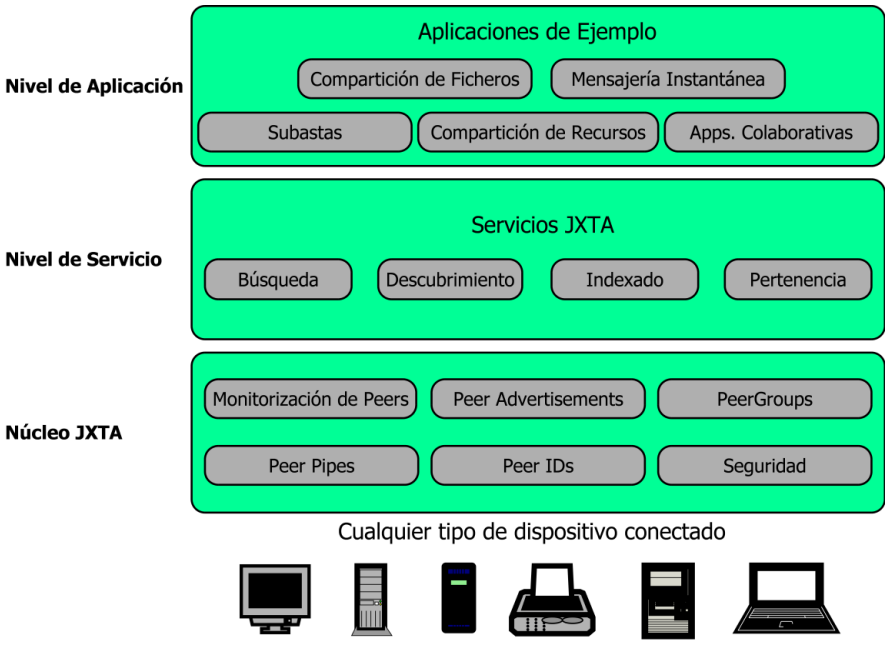


Figura 3: Arquitectura de la red JXTA

- **Núcleo *JXTA*:** Esta capa encapsula las primitivas mínimas y esenciales que son comunes a todas las conexiones *P2P*. Incluye los bloques constituyentes que permiten mecanismos clave para cualquier aplicación *P2P*, como descubrimiento de *peers*, transporte para las comunicaciones (como atravesar *NAT* y *firewalls*), creación de *peers* y grupos de *peers*, y primitivas asociadas con la seguridad.

- **Capa de Servicio:** Esta capa incluye servicios de red que tal vez no son absolutamente necesarios para que una red *P2P* pueda funcionar, pero que son comunes o deseables para cualquier entorno *P2P*. Algunos ejemplos de servicios de red son sistemas de búsqueda e indexado, directorio y almacenamiento, servicios de compartición de ficheros, agrupación de contenidos, traducción de protocolos, autenticación, y servicios de *PKI* (*Public Key Infraestructure*).
- **Capa de aplicación:** Esta capa incluye la implementación de aplicaciones integradas, como mensajería instantánea *P2P*, compartición de documentos y recursos, sistemas de *e-mail P2P*, sistemas de subastas distribuidas, etc.

El límite entre las capas de servicios y de aplicación es flexible. La aplicación de un usuario puede ser vista como un servicio por otro. El sistema completo está diseñado para ser modular, permitiendo a los desarrolladores elegir y utilizar un grupo de servicios y aplicaciones que se ajusten a sus necesidades.

Existen cuatro aspectos fundamentales de la arquitectura *JXTA* que lo hacen diferente de otros modelos de redes distribuidas:

- El uso de documentos *XML* para describir los recursos de red.
- Abstracción de *pipes* a *peers* y de *peers* a *endpoints*, sin depender de una autoridad central de nombrado/direccionamiento como el *DNS*.
- Un esquema uniforme de direccionamiento de *peers* (*ID*).
- Una infraestructura de búsqueda descentralizada basada en *DHT* (*Distributed Hash Table*) para indexado de recursos.

2.2.4. Protocolos

JXTA cuenta con un conjunto de seis protocolos [22] que han sido específicamente diseñados para computación en redes *P2P*. El uso de estos protocolos permite a los nodos cooperar para formar grupos autoorganizados y autoconfigurados independientemente de sus posiciones dentro de la red (*firewalls*, *NAT*, etc.) y sin la necesidad de una infraestructura centralizada que lo gestione todo.

Estos protocolos están diseñados para tener poca carga, hacer pocas suposiciones sobre el nivel de transporte de la red que hay debajo e imponer pocos requisitos al entorno del *peer*, y aún así poder ser utilizados para desarrollar una amplia variedad de aplicaciones y servicios *P2P* en un entorno de red cambiante e inestable.

Los *peers* utilizan los protocolos de *JXTA* para anunciar sus recursos y para descubrir nuevos recursos de red (servicios, *pipes*, etc.) disponibles en otros nodos. Los *peers* crean grupos y se unen a ellos para crear relaciones especiales. También contribuyen al direccionamiento de mensajes permitiendo así una interconexión total entre nodos. Estos protocolos permiten a los *peers* comunicarse sin la necesidad de

entender o gestionar las topologías complejas de red y dinámicas tan comunes hoy en día.

Los protocolos *JXTA* permiten a los *peers* guiar mensajes dinámicamente a través de múltiples saltos en la red a cualquier destinatario (incluso atravesando *firewalls*). Cada mensaje lleva una lista parcial o completamente ordenada con todos los *peers* a través de los cuales podría ser enviado. Los *peers* intermedios pueden ayudar en el guiado de los mensajes utilizando rutas que conocen para acortar y optimizar la ruta que el mensaje tomará. Los protocolos son los siguientes:

- ***Peer Resolver Protocol* (*PRP*)** es el mecanismo por el cual un *peer* puede enviar peticiones a uno o más nodos, y recibir una respuesta (o varias) a su petición. El *PRP* implementa un protocolo de comunicación básica que sigue un esquema de petición/respuesta. El mensaje de respuesta debe coincidir con la petición a través de un identificador único contenido en el cuerpo del mensaje. Las peticiones podrán ser dirigidas a la red entera o a un subconjunto de sus *peers*.
- ***Peer Discovery Protocol* (*PDP*)** es el mecanismo por el cual un *peer* puede anunciar sus propios recursos, y descubrir los recursos publicados por otros nodos (grupos, servicios, *pipes* y otros *peers*). Cada recurso de la red es descrito y publicado utilizando un anuncio. Los anuncios son estructuras de metadatos no sujetas a ningún lenguaje de programación que describen los recursos de red.
- ***Peer Information Protocol* (*PIP*)** es el mecanismo por el cual un *peer* recibe información del estado de otros nodos de la red. Esto incluye estado, tiempo de conexión, carga de tráfico, capacidades y otro tipo de información.

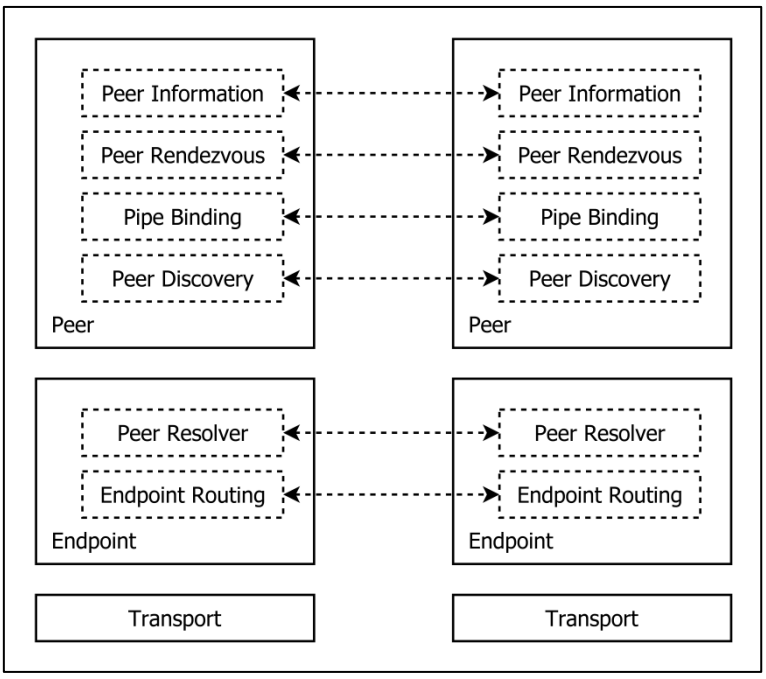


Figura 4: Pila de Protocolos JXTA

- ***Pipe Binding Protocol* (*PBP*)** es el mecanismo por el cual un *peer* puede establecer un canal virtual de comunicación o tubería (*pipe*) entre uno o más nodos. Los *peers* utilizar el *PBP* para unir dos o más terminaciones de una conexión (*pipe endpoints*). Las tuberías proporcionan el mecanismo fundamental de las comunicaciones entre *peers*.
- ***Enpoint Routing Protocol* (*ERP*)** es el mecanismo por el cual un *peer* puede descubrir una ruta (secuencia de saltos) utilizados para poder enviar un mensaje a otro *peer* de la red. Si un *peer* "A" quiere enviar un mensaje a un *peer* "C", y no hay ninguna ruta directa conocida entre ambos, "A" necesita encontrar nodos intermedios que dirijan el mensaje hasta "C". *ERP* es utilizado para determinar la información de la ruta. Si la topología de red cambia y deja inservible la ruta anterior los *peers* pueden utilizar *ERP* para encontrar una ruta alternativa.
- ***Rendezvous Protocol* (*RVP*)** es el mecanismo por el cual los *peers* pueden suscribirse o darse de baja de un servicio de propagación. Dentro de un grupo, los *peers* pueden ser de tipo *rendezvous* o *peers* que escuchan a *rendezvous*. El *RVP* permite enviar a un *peer* mensajes a todas las instancias que estén escuchando dicho servicio. El *Rendezvous Protocol* es utilizado por *PRP* y *PBP* para propagar mensajes.

Esta es una descripción general del funcionamiento de los protocolos de *JXTA*, pero se pueden clasificar de igual forma en función de lo que proporciona cada uno de ellos a cada aplicación *JXTA*. Es un listado muy básico pero que permite hacerse una idea de lo que hacen realmente estos protocolos:

- *Peer Discovery* – Búsqueda de recursos.
- *Peer Resolver* – Servicio genérico de peticiones.
- *Peer Information* – Monitorización.
- *Peer Membership* – Seguridad.
- *Pipe Binding* – Mensajería direccionable.
- *Rendezvous* – Propagación de mensajes.
- *Peer Endpoint* – Encaminamiento.

2.3. Sistemas de reputación

2.3.1. Introducción

Los mercados electrónicos, las aplicaciones *Peer-to-Peer* distribuidas y muchas otras formas de colaboración *online* están basados en la confianza mutua, que empuja a los usuarios que interactúan, a asumir el riesgo inherente que conlleva el realizar transacciones con desconocidos. Los sistemas de reputación se encargan de proporcionar alimentación para sistemas de cálculo de la confianza como predicciones de futuras acciones para un determinado usuario, basándose en comportamientos pasados del mismo. La información de estas acciones pasadas puede ser obtenida de otros usuarios que hayan efectuado alguna transacción con el *peer* del que queremos

conocer su reputación. Sin embargo, la credibilidad de la información recibida de estas terceras personas también debe ser especialmente evaluada.

La meta básica de cualquier sistema de reputación es predecir las acciones futuras de un *peer*, basándose en sus comportamiento pasado. El conocimiento de este comportamiento previo idealmente está basado en la interacción directa con todos y cada uno de los *peers* de la red, incluidos los maliciosos. Para reducir el coste que esto supone, los *peers* comparten sus experiencias personales a través de los sistemas de reputación, lo que hace que se puedan detectar con efectividad los usuarios con mal comportamiento. Esta información compartida se conoce como recomendaciones.

Los sistemas de reputación representan mecanismos suaves de seguridad (*soft security*) que complementan a los mecanismos tradicionales de protección de la información. De esta forma tendríamos dos mecanismos de seguridad: los duros (*hard security*) compuestos principalmente por sistemas de autenticación y control de acceso; y los suaves (*soft security*), también llamados mecanismos de control social.

El concepto de reputación no debe ser confundido con el de confianza, aunque en este caso tienen bastante en común. Así, si se consulta el diccionario *Oxford* de la lengua, mientras la reputación puede ser vista como *"lo que generalmente se dice o se cree acerca del estatus o el carácter de una persona"*, la confianza sin embargo es *"el grado de disposición que tiene un individuo determinado a depender de otro, dada una situación determinada de aparente seguridad, incluso cuando sean posibles consecuencias negativas"*.

De esta forma, la reputación puede considerarse como una medida del grado de confianza (en el sentido de fiabilidad) basada en las referencias o calificaciones de los miembros de la comunidad en la que tiene sentido dicha reputación.

Como se verá en el apartado siguiente, los sistemas de gestión de la reputación distribuidos suponen un buen punto de partida para su integración con las redes *P2P*. De esta forma estarán unidas las tres tecnologías vistas hasta ahora, es decir, redes *P2P*, la plataforma *JXTA* y los sistemas de reputación distribuidos. Esta terna de tecnologías será la base para el sistema de gestión de reputación sobre la que trata este proyecto.

2.3.2. Arquitecturas

La arquitectura de red determina la forma en que las calificaciones y los valores de las reputaciones son comunicados entre los participantes del sistema de reputación. Los dos tipos principales son arquitecturas centralizadas y arquitecturas distribuidas:

- **Sistemas de Reputación Centralizados.**

En este tipo de sistemas, la información sobre el rendimiento de un determinado participante es recogida como el conjunto de las calificaciones obtenidas del resto de miembros de la red que han llevado a cabo alguna transacción directa con dicho

participante. La autoridad central, también llamada centro de reputación es la que se encarga de almacenar todas estas calificaciones, resumirlas en un valor para cada participante, y hacer públicas dichas reputaciones. De esta forma, los participantes pueden utilizar estas puntuaciones enviadas por el sistema central para decidir si interactuar o no con un determinado miembro de la red. La idea es que las transacciones con miembros de la red que tengan buena reputación tendrán más probabilidades de resultar satisfactorias que aquellas realizadas con miembros que tengan una mala reputación.

Tras cada transacción, los miembros que han participado en ella envían puntuaciones acerca del rendimiento mostrado por el otro. El centro de reputación se encarga de recoger todas estas puntuaciones y continuamente actualiza la reputación de estos miembros como una función de las puntuaciones recibidas. Estos valores actualizados estarán disponibles *online* para todos los miembros de la comunidad, de forma que cualquier participante pueda disponer de una información actualizada que le permita decidir si actuar o no con un determinado miembro.

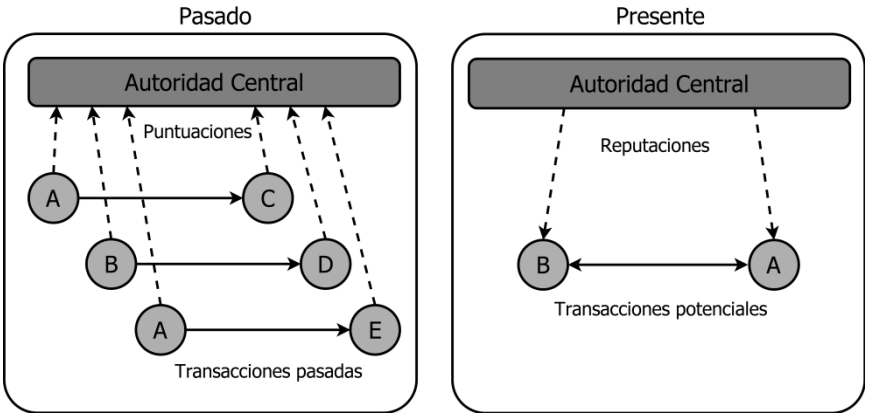


Figura 5: Funcionamiento de un Sistema Centralizado.

Los dos aspectos fundamentales de los sistemas de reputación centralizados son:

Por un lado, protocolos de comunicación centralizados. Estos permiten a los participantes enviar calificaciones de las transacciones realizadas a la autoridad central, así como recuperar de estas reputaciones de miembros con los que realizar futuras transacciones.

Por otro, un mecanismo de cálculo de reputaciones, utilizado por la autoridad central para calcular los valores de reputación de cada participante, basado principalmente en las calificaciones obtenidas, aunque puede depender de algún otro tipo de información.

- **Sistemas de Reputación Distribuidos.**

Existen ciertos entornos en los que un sistema de reputación distribuido es más indicado que un sistema centralizado. En un sistema distribuido no hay ninguna autoridad central a la que enviar las calificaciones o de la que obtener reputaciones de

otros miembros de la red. En lugar de esto, puede haber varios almacenes a los que las calificaciones pueden ser enviadas, o incluso puede darse que cada participante guarde un registro de la opinión sobre cada transacción particular con otros participantes, y distribuye la información de dichos registros a cualquier integrante de la red previa solicitud de la misma.

Un participante de confianza que esté considerando realizar una transacción con otro deberá encontrar estos puntos de almacenamiento de reputación, o intentar obtener puntuaciones sobre ese segundo participante de tantos miembros como pueda de la red, siempre que hayan tenido experiencia directa con él. Este participante de confianza calcula la reputación basándose en las calificaciones obtenidas y, en caso de haber tenido alguna experiencia directa con el miembro en cuestión, la información de dicho encuentro previo será tratada como información privada, y muy probablemente tenga un peso mayor que las calificaciones recibidas.

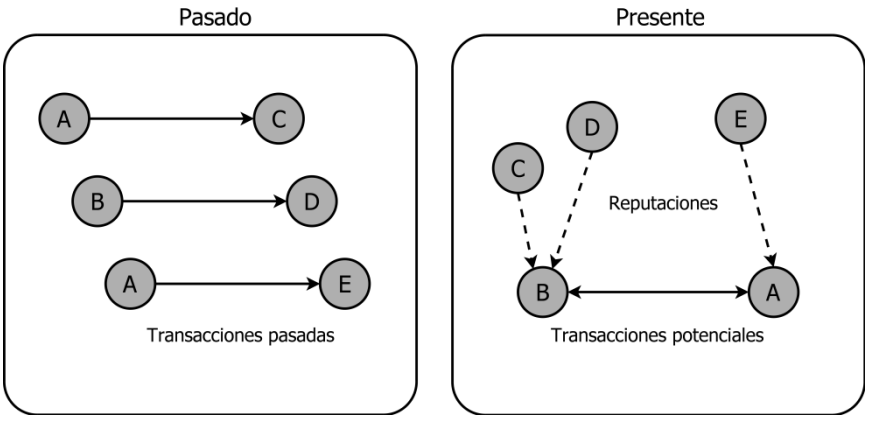


Figura 6: Funcionamiento de un Sistema Distribuido.

Los dos aspectos fundamentales de un sistema de reputación distribuido son:

Un protocolo de comunicación distribuido: Este permitirá a los participantes de la red obtener calificaciones de otros miembros sin la necesidad de contar con ninguna autoridad centralizada.

Un método distribuido de calcular la reputación: Esté será utilizado individualmente por cada integrante de la red para obtener reputaciones de otros miembros basándose en las calificaciones recibidas.

Las redes *P2P* representan un entorno propicio para la gestión distribuida de la reputación. En este tipo de redes, cada nodo tiene los papeles de cliente y servidor, y por lo tanto suele ser denominado con el término *servent* (server + client). Este mecanismo permite a los usuarios superar el típico rol pasivo de la navegación *web* e involucrarse con un papel activo proporcionando sus propios recursos.

2.3.3. Ataques y defensas

Como cualquier otra aplicación o sistema de la red, los sistemas de reputación también pueden ser vulnerables a ciertos tipos de ataques. Para ver los posibles ataques que pueden ocurrir, primero debemos saber qué características tiene el supuesto atacante. Esto incluye si el atacante está dentro o fuera de la red, si es un único atacante o si forma parte de un grupo malicioso, e incluso si el atacante es activo o pasivo.

La naturaleza abierta de los sistemas de reputación nos lleva a asumir que la mayoría de las veces los atacantes pertenecen al propio sistema, es decir, son entidades que tienen acceso legítimo al sistema y que pueden participar en él de acuerdo con sus especificaciones.

También podemos asumir que los atacantes pueden actuar debido a egoísmo, o simplemente por malicia. Los primeros tratan de modificar valores de reputación para su propio beneficio, mientras que los segundos simplemente intentar degradar la reputación del resto de participantes o incluso atentar contra la disponibilidad del sistema.

Como hemos dicho anteriormente, los atacantes pueden actuar solos o formar parte de agrupaciones maliciosas. Aunque ambos escenarios son plausibles, nos centramos en la presencia de grupos formados por atacantes coordinados ya que esos son más difíciles de detectar.

Por último también consideramos más probables los ataques activos, ya que cualquier ataque que se precie sobre un sistema de reputación requiere interacción con el sistema, ya sea introducir información falsificada, modificar información verdadera, negarse a retransmitir información, incumplir los algoritmos de cálculo de las reputaciones, o interrumpir activamente la disponibilidad del sistema.

La clasificación de los ataques contra sistemas de reputación se basa en los objetivos que son atacados dentro de los propios sistemas. La meta de un sistema de reputación, como vimos, es asegurar que los valores de reputación reflejen fielmente las acciones llevadas a cabo por los participantes y que estas no puedan ser modificadas de forma maliciosa. Este objetivo no se podrá cumplir si los usuarios maliciosos pueden incrementar su reputación falsificándola, o si son capaces de devaluar la reputación de otros *peers*. Así, podemos identificar los siguientes tipos de ataques:

- Auto promoción.
- De guante blanco.
- Difamación.
- Orquestados.
- Denegación de servicios.

Los ataques basados en la autopromoción consisten en que los atacantes intentan incrementar su reputación de forma fraudulenta. Estos ataques son posibles únicamente en sistemas que tengan calificaciones positivas exclusivamente, por lo que se puede considerar un ataque contra la formulación del cálculo de reputaciones del sistema.

Los ataques de guante blanco son aquellos en los que los atacantes abusan del sistema dejando que su reputación se degrade poco a poco, y luego escapando de las consecuencias de su mala práctica restaurando su reputación a través de alguna debilidad del sistema. A menudo estos atacantes intentan acceder de nuevo al sistema utilizando otras identidades, por lo que serán más vulnerables aquellos sistemas en los que sea fácil cambiar de identificadores de usuario. Estos ataques también van contra la formulación y se ceban en sistemas en los que las calificaciones sean exclusivamente negativas, así los nuevos miembros del sistema tendrán igual reputación que aquellos que se comportan correctamente.

En los ataques de difamación, una o más entidades producen calificaciones negativas sobre otros usuarios. Como ocurría con los ataques de auto promoción, los sistemas sin un sistema de autenticación son muy vulnerables a este tipo de ataques.

A diferencia de los ataques anteriores, en los ataques orquestados los atacantes tratan de coordinar varios tipos de ataques existentes. En estos ataques los malhechores pueden cambiar de comportamiento a lo largo de tiempo o repartirse los objetivos a los que atacar, de forma que es más difícil detectarlos. Este tipo de ataques atenta contra la formulación del sistema, y si los atacantes son suficientemente numerosos para poder distribuir sus propias calificaciones, estos pueden llegar a alterar los valores de reputación en su propio beneficio.

Por último, los ataques de denegación de servicio buscan interrumpir el mecanismo de funcionamiento del propio sistema de reputación. Los sistemas con un alto grado de centralización o aquellos en los que la redundancia de la información no es suficiente, son muy vulnerables a este tipo de ataques, ya que los atacantes buscan la sobrecarga de las autoridades centrales del sistema.

Una vez descritos los mecanismos de ataque contra los sistemas de reputación, debemos mencionar también aquellas estrategias que permiten defenderlos. Aunque no existe ningún mecanismo que permita defenderse de todos los tipos de ataques simultáneamente, la mayoría de las defensas que veremos a continuación son efectivas contra atacantes egoístas y contra algunos ataques llevados a cabo por agrupaciones maliciosas.

El primero de estos mecanismos es prevenir la utilización de múltiples identidades por parte de un usuario (*Sybil attacks*). En un sistema centralizado podemos evitar este tipo de ataques haciendo que una entidad central se encargue de verificar las credenciales de cada usuario, y que estas sean únicas o que para obtenerlas se requiera algún tipo de coste monetario o computacional para el posible atacante. En los sistemas

distribuidos es algo más complicado, pero se puede conseguir una defensa bastante buena utilizando identificadores únicos como por ejemplo claves públicas en base a direcciones *IP*, o utilizando entidades coordinadoras que detecten nodos con múltiples identidades.

Otro método para evitar ataques es minimizar la creación de falsos rumores. Para poder hacer esto algunos sistemas proponen integrar en el sistema algún tipo de contabilidad a través de firmas digitales o pruebas irrefutables, que pueden ser implementadas como mecanismos criptográficos que avalen las transacciones que se llevan a cabo en el sistema.

Si esto no funciona también se puede recurrir a la disminución de la propagación de estos falsos rumores. Si se basase el cálculo de las reputaciones en las interacciones directas entre los *peers* esto sería muy fácil de evitar, pero ya vimos que las transacciones directas entre todos y cada uno de los nodos del sistema es prácticamente inviable. Una solución propuesta por el algoritmo *EigenTrust* [23] es la presencia de nodos cuya fiabilidad está establecida de antemano y que serán los responsables de mitigar el impacto de estos falsos rumores. El problema de este sistema es el daño que se puede ocasionar hasta que los nodos comunes sean capaces de localizar a los fiables.

Otro método consiste en prevenir los abusos del sistema a corto plazo, en los que los nodos dejan que su reputación se degrade para entrar más tarde con una identidad nueva. Esto se puede lograr haciendo que los miembros que se incorporen por primera vez tengan que ganarse la confianza y obligándoles a comportarse correctamente durante un tiempo hasta que su reputación se incremente.

Por último está la defensa contra los ataques de denegación de servicio, que depende de la estructura empleada por el sistema para el almacenamiento y la distribución de los valores de reputación. Para conseguirlo se pueden utilizar sistemas como el de *TrustMe* [24] en los que los nodos que se encargan del cálculo y la propagación de los valores de reputación son elegidos aleatoriamente.

2.3.4. Ejemplos prácticos

Hasta este punto se ha intentado explicar la filosofía y la teoría subyacente de los sistemas de gestión de la confianza y la reputación, pero esto no es de mucha ayuda si no se conoce realmente para qué sirven. Es el momento de dar un paso más y presentar algunos de los trabajos que se centran en estos sistemas de confianza, de esta forma se verá su utilidad.

Los ejemplos que se verán a continuación se pueden dividir en dos grandes categorías: los trabajos de investigación y las implementaciones prácticas.

En primer lugar se hará un repaso por varias de las tecnologías sobre las que se está trabajando para hacer uso de los sistemas de reputación o confianza. Son

principalmente trabajos teóricos que establecen buenas bases para trabajos futuros. Entre ellos se pueden destacar:

- ***CORE*** [25]

  La motivación del sistema *CORE* se basa en la necesidad de prevenir los comportamientos maliciosos y egoístas por parte de los nodos en las redes móviles y *ad-hoc* (*MANET* por sus siglas en inglés). Aquí, el valor de reputación final es una combinación del comportamiento observado directamente de un *peer* y las observaciones indirectas compartidas por terceros, que serán evaluados en el rango [-1, +1] donde se le asigna la mayor puntuación a las operaciones satisfactorias. La propagación de estas observaciones no se hace de forma directa, de forma que si un nodo desea conocer la reputación de un *peer* dado, deberá realizar peticiones explícitas a otros nodos para conseguirla.

  Los mecanismos de defensa del sistema *CORE* se centran principalmente en los comportamientos maliciosos contra el cálculo y la distribución de las reputaciones. Este sistema no cuenta con ningún mecanismo para preservar la integridad de los contenidos propagados por la red.

- ***EigenTrust*** [23]

  El sistema de reputación *EigenTrust* surgió de la necesidad de filtrar y eliminar contenidos falsificados en las redes *P2P* de compartición de ficheros. Mediante *EigenTrust* se calcula una reputación global para cada *peer* dentro del sistema basado en las opiniones locales del resto de *peers* de la red. Estas opiniones locales se agrupan en un formato de matriz y los valores globales de reputación se obtienen a partir de ella.

  Los mecanismos de defensa de este sistema se basan en dar mayor credibilidad a la información recibida de aquellos nodos con los que se ha interaccionado previamente para minimizar el impacto de los nodos maliciosos. También se emplean nodos cuya confianza se presupone para hacer llegar la información correcta de reputación a nodos cuyo comportamiento sea el correcto, para asegurar que dicha información es, en efecto, la que se propaga.

- **Poblano** [26]

  Es un sistema de gestión de la confianza distribuido en el que cada nodo almacena su experiencia personal en base a las interacciones pasadas con otros *peers*. No existe un valor único de confianza ya que cada nodo, para calcular una reputación combina su experiencia personal con los valores de confianza obtenidos de otros nodos, dando siempre más peso a la primera. Los valores de confianza se distribuyen en base a un protocolo de petición respuesta, y a cada interacción que se lleva a cabo entre dos *peers* cada nodo actualiza el valor de

reputación del otro. Este será el sistema en el que se basará principalmente el sistema de gestión de reputación que se implementará en este proyecto.

Como se ha mencionado antes, la base teórica queda huérfana si no se respalda con ejemplos que se puedan encontrar en la vida cotidiana, es decir, tecnologías con las que un usuario pueda interactuar en su vida normal. En este ámbito se pueden mencionar los siguientes ejemplos:

- **eBay** [27]

  Este es un popular sitio *Web* de subastas que permite a los vendedores poner sus productos en venta y a los compradores pujar por dichos productos. El mecanismo de *feedback* empleado por *eBay* permite a compradores y vendedores evaluarse mutuamente, dando puntuaciones positivas, negativas o neutrales (-1, 0, 1) tras completar las transacciones.

  El sistema de *feedback* de *eBay* es un sistema centralizado, en el que un órgano central recoge todas las evaluaciones y calcula las reputaciones. Este valor total de reputación es la suma de todos los votos positivos menos los negativos en un rango de tiempo determinado.

- **Amazon** [28]

  Amazon es principalmente un sitio *Web* de venta de libros que permite a los usuarios escribir opiniones de libros. Cualquier persona puede convertirse en miembro de Amazon simplemente con registrarse, y las opiniones consisten en el texto de la opinión propiamente dicho y una puntuación de una a cinco estrellas. La media de todas las puntuaciones da la reputación del libro. A su vez, las opiniones son evaluadas como "útil" o "no útil", y todas estas evaluaciones se mostrarán junto con la opinión. Mediante estas evaluaciones Amazon establece la reputación del opinante.

- **Google** [29]

  El sistema de clasificación de *Google* supone un avance respecto de los antiguos buscadores. En estos viejos buscadores, como *Altavista*, simplemente se mostraban todas las páginas *Web* que coincidían con la búsqueda del usuario, incluso aquellas inútiles. Para contrarrestar esto *Altavista* propuso un sistema basado en lógica binaria para la combinación de palabras clave. No resultó una buena solución.

  PageRank es un conjunto de algoritmos definidos por *Google* que representa un método de clasificación de páginas *Web* en función de su reputación. Esto es, por decirlo de alguna forma, que *PageRank* clasifica a una página en función del número de páginas *Web* que apuntan a la primera. Esto puede ser descrito como un sistema de reputación, ya que la colección de los hiperenlaces a una

determinada página puede ser vista como información pública que puede ser combinada para determinar un valor de reputación.

2.3.5. Poblano

En este punto se debe hacer una mención especial a *Poblano*, que como se mencionó en el apartado anterior, es un modelo de gestión de la confianza diseñado por *Sun* específicamente para *JXTA*. Por tanto, el sistema desarrollado se basa en él, para hacer las extensiones y adaptaciones necesarias.

Por este motivo, se va a dedicar este apartado a la presentación de dicho modelo, lo que ayudará a comprender mejor el diseño especificado en los capítulos de descripción e implementación del sistema.

2.3.5.1. Representación de relaciones de confianza

En este subapartado se intentarán dar los primeros pasos hacia la implementación de un modelo que permita la representación de la confianza en redes *P2P* distribuidas. Para que este sistema se pueda implementar se debe abordar en términos cercanos a la programación, asunto que se tratará de lleno en el apartado que describe las estructuras y los módulos en el capítulo de descripción del sistema. En este subapartado se intentará dar una visión de los mismos conceptos desde un punto de vista menos formal.

En los actuales sistemas de gestión de la confianza, como los utilizados en los proyectos *Free Haven* [30] o *Publius* [31], el **grado de confianza** se calcula en base a parámetros como rendimiento, honestidad, fiabilidad, etc., referentes a **un *peer*** desde el punto de vista de otros. Por ejemplo, si un *peer* hace trampas jugando a las cartas, automáticamente le será prohibida la entrada a una nueva partida.

Por el contrario, para un grupo de personas interesados en cocinar, las medidas anteriores estarán demasiado condicionadas por los riesgos personales, en lugar de estarlo por los contenidos que estos comparten, por lo que será de poca utilidad. De hecho, en el caso del grupo de cocina, la confianza deberá estar condicionada por la importancia de los datos o la calidad de las recetas compartidas. En este sistema la confianza tiene diferentes componentes o factores, por lo que se busca que estos valores de confianza estén basados en los intereses del grupo y en la calidad (reputación) de los contenidos que se comparten en él.

Para poder recopilar toda la información sobre los intereses de un grupo, este sistema introducirá un **componente de confianza en contenido**, que se complementará con el de **confianza en *peer***. Estos componentes se explicarán más adelante, pero se pueden ver en la figura 7.

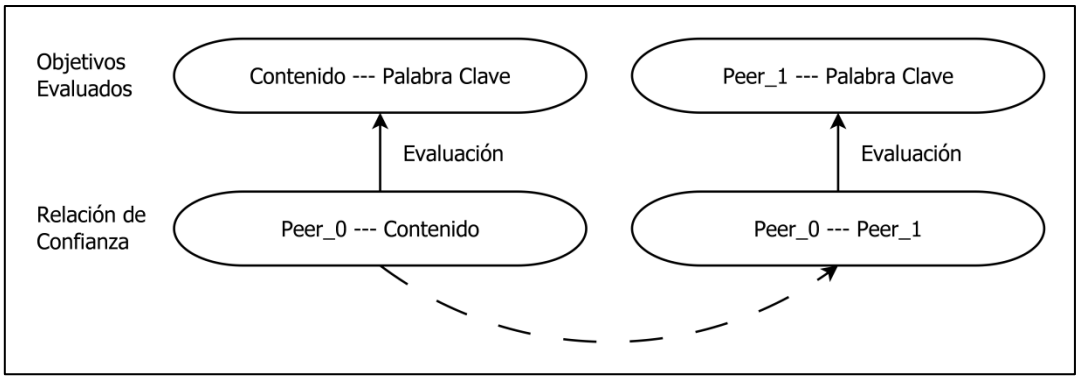


Figura 7: Relaciones de Confianza

Para comenzar, un usuario (*Peer_0*) necesita evaluar su confianza en un contenido para construir su relación de confianza con dicho contenido. Para evaluar esta confianza con respecto a los intereses del *peer*, cada contenido debe ser tenido en cuenta en función de una palabra clave a la que hace referencia. Así, el resultado de esta evaluación de todos los contenidos que *Peer_0* recibe de otro usuario *Peer_1* será utilizado para evaluar la confianza de *Peer_0* en *Peer_1*.

De esta forma, la evaluación de la confianza en un *peer* estará basada en los intereses del usuario o, dicho de otro modo, en las palabras clave en las que está interesado. En este caso, el componente de confianza en contenidos es distinto del concepto tradicional de confianza, que podríamos identificar con el factor de riesgo en nuestro sistema. Este factor de riesgo sí que dependerá de factores relacionados con el usuario remoto propiamente dicho, como por ejemplo, su rendimiento.

Para representar los diferentes componentes de la confianza en nuestro sistema se emplearán las siguientes estructuras:

- **Confianza en contenido:** Este parámetro indica la confianza que un *peer* tiene en un contenido determinado en base a una palabra clave concreta. Así, para identificar correctamente uno de estos valores de confianza se deberá conocer básicamente dicha palabra clave, un identificador de dicho contenido y el valor de confianza propiamente dicho. Este valor de confianza podrá tomar valores numéricos enteros que van desde -1 (total desconfianza) hasta 4 (total confianza).
- **Confianza en *peer*:** Esta parámetro indica la confianza que el *peer* local tiene en un *peer* determinado, al igual que ocurría en el caso anterior, en relación con una palabra clave determinada. De esta forma, para identificar unívocamente uno de estos valores se deberá conocer la palabra clave, el identificador del *peer* y el valor de confianza en sí. Este valor de confianza podrá tomar valores numéricos enteros que van desde -1 (total desconfianza) hasta 4 (total confianza).
- **Factor de Riesgo:** Este parámetro, como se decía anteriormente, mide el riesgo que entraña la comunicación con un *peer* determinado. Este factor

quedará determinado si se conoce el identificador del *peer* al que hace referencia y otros factores como el rendimiento y el grado de accesibilidad de dicho *peer*. El riesgo podrá tomar valores numéricos enteros que van desde 0 (mínimo riesgo en la interacción) hasta 4 (máximo riesgo).

Tanto los valores de confianza en contenidos como los de confianza en *peer* tendrán una doble medida para representar las confianzas propiamente dichas. Por un lado estarán compuestas por el valor de confianza en sí, que ya hemos visto, pero por otro lado se incluirá en todas las medidas un grado de popularidad. Esta popularidad indicará el número de veces que se haya accedido al contenido o al *peer* en cuestión.

De estos tres componentes, los más interesantes son la confianza en *peer* y el factor de riesgo. A través de ellos se podrá determinar si el *peer* objetivo de una petición estará dispuesto a cooperar y, por lo tanto, si será digno de confianza.

Por otro lado, la confianza en contenido se puede considerar como la base para el cálculo de la confianza en un *peer*, ya que, en general, para un usuario es más fácil evaluar un contenido que evaluar directamente a un *peer*.

2.3.5.2. Tablas de confianza

En este subapartado nos centraremos en la parte local del sistema, realizando una descripción detallada de todas las tablas que deben estar almacenadas en memoria (y que serán guardadas en disco entre ejecuciones) y el formato que deberán tener para que todo funcione correctamente.

- **Tabla de confianza en contenidos**

Desde el punto de vista del usuario, esta tabla se puede ver a su vez como un conjunto de otras tablas más pequeñas anidadas, cada una de ellas correspondiente a un grupo determinado al que pertenezca dicho *peer*. De esta forma tendremos los datos aislados y organizados por estos grupos para facilitar la búsqueda de valores de confianza en contenidos. Dependiendo del grupo en el que se realice la búsqueda se utilizará una tabla u otra.

Las tablas de cada uno de los grupos, a su vez, están formadas por otras más pequeñas que organizarán los valores de confianza en contenidos según las palabras clave a las que hagan referencia. De esta forma existirán tantas tablas secundarias como palabras clave conozca el *peer*. Un esquema de esta jerarquía de tablas para representar la confianza en contenidos puede verse en el siguiente esquema:

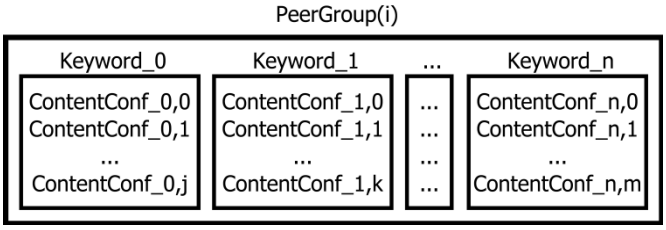


Figura 8: Tabla de confianza en contenidos para un grupo i

En este caso, para el grupo "i" estarán registrados contenidos que tienen relación con "n" palabras clave diferentes, y para cada una de estas palabras, a su vez, habrá un conjunto de valores de confianza en contenidos *ContentConf_x,y*. Esta notación quiere decir: valor de confianza correspondiente al contenido "y" si se evalúa desde el punto de vista de la palabra clave "x".

En cualquier caso, cada uno de estos valores *ContentConf_x,y* no se representa exclusivamente con un valor numérico sino que, tal y como se ha propuesto en apartados anteriores, está compuesto por otros valores secundarios que definen por completo un valor de confianza en contenido. Estos valores son el nombre y el identificador del contenido, la popularidad del mismo, la marca de tiempo que indica cuándo se creó dicho valor de confianza y, cómo no, el valor de confianza propiamente dicho.

Esta es la información correspondiente a un único grupo dentro del sistema, lo que supone una cantidad de información bastante alta. Si a esto añadimos el número de grupos a los que pertenece el *peer* podemos encontrarnos con un volumen de datos inmanejable si no estuviese organizado de esta forma.

- **Tabla de confianza en *peers* dependiente de grupo.**

Esta tabla tiene muchos aspectos en común con la de contenidos. Desde el punto de vista del usuario, al igual que ocurría antes, no será una única tabla, sino que serán múltiples tablas, una por cada grupo al que pertenezca el *peer*. En este caso podemos ver que el número de tablas de grupo para la confianza en contenidos será el mismo que el de la confianza en *peers*.

Del mismo modo, cada tabla de grupo estará formada por otras tablas más específicas, cada una de las cuales estará ligada a una palabra clave, que será por la que se esté evaluando a cada *peer*. Se puede ver esta estructura en la figura número 9:

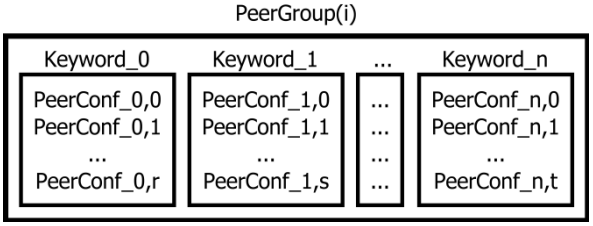


Figura 9: Tabla de confianza en peer para un grupo i

En este caso, para el grupo "i" se tendrá en los registros tablas para "n" palabras clave diferentes. Cada una de estas tablas estará formada por un conjunto de valores *PeerConf_x,y* que representarán un valor de confianza. Así, *PeerConf_x,y* quiere decir: el valor de confianza correspondiente al *peer* "y" si se está considerando respecto a la palabra clave "x".

Al igual que ocurría con las entradas de las tablas de confianza en contenido *ContentConf_x,y*, las entradas de estas tablas *PeerConf_x,y* tampoco estarán formadas por un único valor numérico, sino que estarán compuestas de varios datos que nos permitirán identificar por completo dicho valor de confianza en *peer*. Estos valores son, nombre e identificador del *peer*, valor decimal numérico que representa la confianza, valor entero que representa su popularidad, marca de tiempo que indica cuando fue creado dicho valor, la suma de los valores de confianza de todos los contenidos obtenidos de este *peer* y el número de contenidos obtenidos del mismo.

- **Tabla de confianza en *peers* independiente de grupo.**

Se puede decir que esta tabla es un caso particular, y más sencillo, de la tabla de confianza en *peers* anterior. Se dice que es un caso particular porque plantea el mismo problema, con la diferencia que habrá una única tabla general, por lo que se puede hacer la analogía con el caso anterior para el número de grupos a los que pertenece el *peer* igual a uno.

Esto es una forma bastante simple de ver las tablas de confianza independientes de grupo, ya que en realidad nos muestran un resumen de toda la información contenida en las tablas de cada uno de los grupos. Esto nos servirá para evaluar a un *peer* determinado en base a los contenidos que ha proporcionado a lo largo del tiempo, independientemente de cada grupo, por lo que nos dará una visión más completa del comportamiento del mismo.

Se puede ver de la siguiente forma: cuando un *peer* del sistema obtiene un contenido de otro usuario de la red, y realiza una evaluación del mismo, modificará la tabla de confianza en contenido para un grupo en concreto, así como la tabla de confianza en *peer* para un grupo en concreto, dejando intactas las del resto de grupos. Por el contrario, cada vez que se lleva a cabo este proceso, siempre se actualizará alguno de los valores de la tabla de confianza en contenidos independiente de grupos. La estructura desde el punto de vista de usuario es la siguiente:

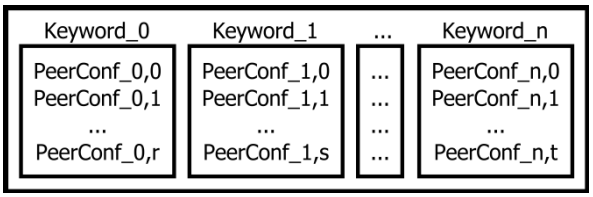


Figura 10: Tabla de confianza en peer Independiente de grupo

Ya se ha hecho referencia en la explicación de la tabla anterior al significado de cada uno de los valores *PeerConf_x,y*. En estas tablas los valores son exactamente los mismos, con los mismos elementos constituyentes, por lo que se puede obviar la explicación para pasar al punto de vista del almacenamiento en disco.

- **Tabla de riesgo**

La última de las tablas que debe contener el sistema es la más sencilla de todas ellas con diferencia, pero no por ello tiene menor importancia. Es la tabla que nos permitirá almacenar los valores con los que se calculará el riesgo de interacción con un *peer* determinado.

Esta tabla no se rige por grupos de intereses ni por palabras clave, aunque tiene que ver con todos ellos. Cada vez que un usuario realiza cualquier interacción con otro nodo de la red se generará una nueva observación que derivará en la actualización de la entrada de esta tabla correspondiente a dicho *peer*, independientemente del grupo o de la palabra clave que se hayan visto envueltos en dicha interacción.

Por lo tanto, esta será una tabla en la que simplemente habrá un conjunto de valores de riesgo, tal y como representa la figura 11, que emplea la misma notación que las anteriores:

Figura 11: Formato Tabla de riesgos

2.4. eXtensible Markup Language

El *Extensible Markup Language*, o de forma abreviada *XML*, es una especificación creada por el *World Wide Web Consortium* (W3C) [32], concretamente por el grupo de trabajo *XML*. Según este grupo, la definición más acertada para el lenguaje *XML* es la siguiente:

"El lenguaje de marcado extensible (XML) es un subconjunto de SGML. El objetivo consiste en posibilitar que el SGML genérico pueda suministrarse, recibirse y procesarse en la Web del mismo modo que lo hace HTML. XML ha sido diseñado para que su implementación sea sencilla y permita interoperar tanto con SGML como con HTML".

Este extracto puede encontrarse en la primera especificación oficial de *XML* [33], la 1.0, creada en 1998 por el grupo de trabajo *XML*, tal y como mencionamos antes.

XML es un lenguaje de marcado, diseñado específicamente para enviar información por la *Web*, exactamente igual que el lenguaje *HTML* (*Hypertext Markup Language*) que ha sido, desde el nacimiento de la *Web*, el lenguaje estándar utilizado para la creación de páginas *web*. A pesar de esto, aunque *XML* no puede considerarse

como un sustituto del *HTML*, sí que permite eliminar alguna de las limitaciones que posee este lenguaje.

La definición de *XML* se basa en el uso de una sintaxis muy escueta. Sin embargo, cuando se crea un documento *XML*, en lugar de tener que utilizar un conjunto limitado de elementos predefinidos (como ocurre con *HTML*), podemos generar nuestros propios elementos asignándoles los nombres que más nos convengan (de aquí el término *extensible* de *XML*). De esta forma, podemos utilizar *XML* para describir virtualmente cualquier tipo de documento, desde una base de datos hasta una partitura musical.

Un documento *XML* está estructurado mediante una jerarquía de tipo árbol, con unos elementos anidados dentro de otros, y con un único elemento de nivel superior, conocido como *elemento raíz* que contiene a todos los demás. De esta forma, podemos emplear *XML* para definir documentos estructurados jerárquicamente.

Tal y como se ha mostrado en las sección correspondiente a *JXTA*, *XML* es la base del conjunto de protocolos presentados por esta plataforma para la integración con redes *Peer-to-Peer*. La utilización de esta tecnología en el sistema de gestión de reputación sobre la que trata esta memoria hace indispensable la presentación de *XML*.

Por otro lado, *XML* será un pilar fundamental de este sistema de gestión de la reputación en lo que a almacenamiento de datos se refiere, tal y como se verá más adelante, ya que todo se basa en la utilización de ficheros con este formato.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <department>
  - <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
  - <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
```

Figura 12: Ejemplo de Código XML

2.4.1. SGML, HTML y XML

SGML [34] (*ISO* 8879) significa *Standard Generalized Markup Language*, y es el predecesor de todos los lenguajes de marcado. Tanto *HTML* como *XML* derivan de este lenguaje, aunque de forma diferente. *SGML* define una sintaxis básica, pero nos permite generar nuestros propios elementos, de ahí el término "generalizado". Para utilizar *SGML* en la descripción de un documento particular, deberemos inventar un conjunto de elementos y una estructura de documento adecuados.

Se denomina aplicación *SGML* a un conjunto de estos elementos de propósito general, utilizado para describir un determinado tipo de documento. Una aplicación *SGML* también incluye reglas que especifican la manera con que los elementos podrán ser organizados. Así podemos definir nuestra propia aplicación *SGML* para que describa un determinado tipo de documento con el que queramos trabajar, o una organización de estándares podría definir una aplicación *SGML*, que describiera un tipo de documento utilizado extensamente. El ejemplo más conocido de este último tipo es el *HTML* [35], que es una aplicación *SGML* desarrollada en 1991 para describir páginas *web*.

SGML parece el lenguaje extensible perfecto para describir documentos *web*, sin embargo, los miembros del *W3C* lo consideraban demasiado complicado para suministrar información en la *Web* de manera eficiente. Según pensaron, lo que hacía falta era un subconjunto de *SGML* diseñado específicamente para proporcionar información dentro de la *Web*. En 1996, el Grupo de Trabajo *XML* del *W3C* desarrolló este subconjunto, al que denominaron lenguaje de marcado extensible, y que fue diseñado para una implementación sencilla.

XML es, por tanto, una versión simplificada de *SGML*, optimizada para la *Web*. Al igual que *SGML*, *XML* nos permite determinar nuestro propio conjunto de elementos a la hora de describir un documento concreto.

En cualquier caso no podemos decir que *XML* pueda sustituir a *HTML*, que es el principal lenguaje utilizado para indicar a un navegador web cómo representar la información. De hecho, *XML* se utiliza de manera conjunta con *HTML* ampliando la capacidad de las páginas web para suministrar cualquier tipo de documento, presentar información altamente estructurada e incluso ordenar, filtrar, localizar y manipular información.

2.4.2. Objetivos

En la especificación [36] de *XML* recogida en la página del *W3C* nos encontramos con un listado en el que se enumeran las 10 metas básicas que pretende alcanzar este lenguaje:

1. ***XML* se debe poder utilizar directamente en Internet.**

Tal y como se muestra en la definición del *W3C*, *XML* fue diseñado principalmente para almacenar y suministrar información a través de la *Web*, por lo que su utilización en Internet se presupone.

2. ***XML* debe soportar una amplia variedad de aplicaciones.**

Aunque su principal objetivo consiste en proporcionar información a través de la *Web* mediante programas de servidor y navegadores, *XML* también está diseñado para ser utilizado con otros tipos de programas; por ejemplo, *XML* ya se emplea para compartir información entre programas financieros, para distribuir y actualizar software y para escribir scripts de voz que puedan enviarse a través del teléfono.

3. ***XML* debe ser compatible con *SGML*.**

Como hemos visto en apartados anteriores, *XML* es un subconjunto de propósito especial de *SGML*. Una de las ventajas de esta funcionalidad es que las herramientas de software de *SGML* se pueden adaptar muy fácilmente para trabajar con *XML*.

4. **Debe ser fácil escribir y crear programas que procesen documentos *XML*.**

Partimos de la base de que *XML* debe ser práctico, por lo tanto, la creación de navegadores y demás programas que procesen documentos *XML* tendrá que ser muy simple. De hecho, la creación de *XML* como un subconjunto de *SGML* se produjo por la complejidad para generar programas que procesaran documentos *SGML*.

5. **El número de funcionalidades opcionales deberá mantenerse en un mínimo absoluto, preferiblemente cero.**

La motivación de este objetivo viene dada por el requisito previo de que sea sencillo crear programas que procesen documentos *XML*, y por lo tanto, la existencia de este número mínimo de funcionalidades lo facilita. La cantidad de funcionalidades opcionales en *SGML* fue una de las causas por las que se le calificó de poco práctico para definir documentos *web*.

6. **Los documentos *XML* deben ser legibles por los humanos y razonablemente claros.**

XML está diseñado para convertirse en lengua básica para el intercambio de información entre los usuarios y los programas de todo el mundo. El que sea inteligible para los humanos permite alcanzar este objetivo, posibilitando a las personas (y a los programas de software especializados) la lectura y escritura de documentos *XML* sin demasiada dificultad. Su legibilidad distingue a *XML* de la mayoría de los formatos propietarios, utilizados en las bases de datos y en los documentos de los procesadores de texto.

7. **El diseño de *XML* deberá prepararse rápidamente.**

XML será un estándar viable únicamente si la comunidad de programadores y usuarios lo adopta. Este estándar necesita por tanto ser completado antes de que esta comunidad comience a adoptar estándares alternativos, los cuales tienden a producir las compañías de software a un ritmo vertiginoso.

8. **El diseño de *XML* deberá ser formal y conciso.**

La especificación de *XML* está escrita en un lenguaje formal, utilizado para definir lenguajes informáticos y que se conoce como notación *EBNF* (*Extended Backus-Naur Form*) [37]. Este lenguaje formal, aunque difícil de leer a primera vista, resuelve las ambigüedades y en último término facilita la creación de documentos *XML* y

especialmente el software de procesamiento de *XML*, alentando aún más la adopción de este lenguaje.

9. Los documentos *XML* deberán ser fáciles de crear.

Para que *XML* se convierta en un lenguaje práctico de marcado para los documentos web, no sólo debe ser fácil la creación de programas de procesamiento *XML*, sino que también los propios documentos *XML* tendrán que poder crearse de forma sencilla.

10. La brevedad en los marcadores *XML* es de mínima importancia.

Para conseguir el objetivo de que los documentos *XML* sean inteligibles para los humanos es preciso este requisito, así que los marcadores *XML* no deberán ser tan concisos que lleguen a convertirse en crípticos.

2.4.3. Aplicaciones y usos

Una aplicación *XML* se define normalmente generando una definición de tipo de documento o *DTD*, que es un componente opcional de un documento *XML*. Una *DTD* es similar al esquema de una base de datos: define y asigna nombres a los elementos que se pueden utilizar en el documento, el orden en que dichos elementos podrán aparecer, los atributos de los elementos que podrán utilizarse y otras funcionalidades del documento. De esta forma, si disponemos de la *DTD* en el documento, estamos restringiendo los elementos y estructuras que podremos utilizar, forzándolo a adaptarse al estándar de la aplicación.

Además de las aplicaciones *XML* para describir las clases de documentos, se han definido diversas aplicaciones *XML*, que podemos emplear dentro de cualquier tipo de documento *XML*. Algunos ejemplos de estas aplicaciones son los siguientes:

- ***XSL*** [38] **(*Extensible Stylesheet Language*):** permite crear hojas de estilo de documentos utilizando la sintaxis *XML*.
- ***XML Schema*** [39]**:** Alternativa al *DTD* que permite escribir esquemas detallados para documentos *XML*.
- ***XLink*** [40] **(*XML Linking Language*):** permite vincular documentos *XML*.

Todo lo que se ha visto hasta ahora hace ver que *XML* es un concepto interesante, y de hecho lo es para el mundo real. Entre las aplicaciones prácticas de *XML* podemos encontrar algunas como:

- Almacenamiento de bases de datos: Utilizando *XML* para etiquetar los campos de información dentro de cada registro.
- Almacenamiento de gráficos vectoriales (*VML*).
- Comunicación entre aplicaciones a través de la *Web*: utilizando mensajes *XML*.

- Formato de fórmulas matemáticas y contenido científico en la *Web* (*MathML*).
- Y un largo etcétera que abarca múltiples campos de aplicación, desde astronomía hasta teología, pasando por documentación legal.

CAPÍTULO 3: DESCRIPCIÓN GENERAL DEL SISTEMA

En primer lugar se deben tratar cuestiones básicas de este sistema que permitirán conocer su estructura y darán una visión completa del mismo. Entre estos temas se puede encontrar desde qué es un *peer* hasta, por ejemplo, el tipo de red que se utilizará o la plataforma en la que se desarrollará su funcionamiento.

Este sistema de gestión de la reputación se basa en la utilización de una red de tipo *P2P* en la que todos los nodos participantes actúan como iguales, intercambiando constantemente informaciones de reputación que lo mantendrán actualizado en todo momento. En lugar de optar por la creación de una red *P2P* específica para este proyecto, se ha estimado utilizar como plataforma básica la red *JXTA* [18], un proyecto iniciado por *Sun Microsystems* para ofrecer un modelo unificado de red *P2P*.

En este punto se debería hablar del concepto de *peer* en este sistema. Un *peer* es cualquier entidad que actúa dentro del sistema de gestión de la reputación compartiendo contenidos locales, buscando otros remotos y evaluándolos a todos ellos, creando así unos valores de reputación. Se puede establecer una analogía entre *peer*, o nodo, y usuario de la red. Así, cada usuario contará con su propia información de reputación que será independiente de la de cualquier otro *peer* de la red, pero que será calculada en conjunto.

Se debe establecer una diferencia clara entre *peer*, o usuario, y dispositivo. Al igual que ocurre con otros sistemas, un dispositivo, como por ejemplo un ordenador personal, puede ser utilizado por varios usuarios de forma indiferente, cada uno de los cuales tendrá su propia información almacenada en carpetas personales. Del mismo modo, un usuario puede conectarse a la red desde distintos dispositivos, sin que ello suponga una pérdida de información o problemas en la localización del mismo.

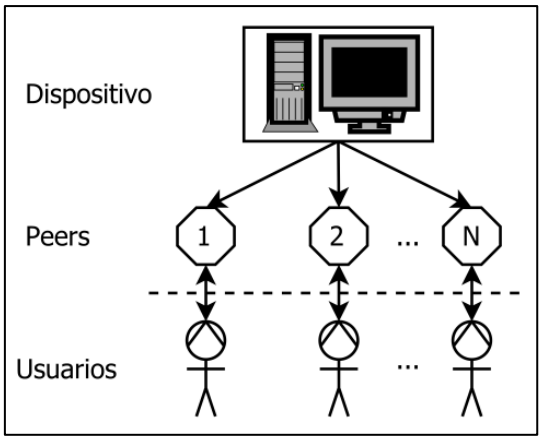


Figura 13: Relación Usuario-Peer

Cada usuario, en su carpeta personal dentro del dispositivo desde el que accede a la red *P2P*, contará con varios ficheros *XML* que contendrán toda la información que ha ido recabando a lo largo de sus sucesivas conexiones, así como otros ficheros que determinarán la identidad de dicho usuario dentro de la red. Por otra parte también

contará con una carpeta en la que se irán almacenando los ficheros de *log* que representarán todas las interacciones que el usuario ha llevado a cabo.

La ventaja que ofrece la plataforma *JXTA*, además de un marco común para la intercomunicación de los usuarios, es la división de la red en grupos. Cada uno de estos grupos puede tomarse como un conjunto de usuarios interesados por los mismos temas, que compartirán contenidos con usuarios afines. Pueden existir tantos grupos como diferentes intereses puedan tener los usuarios y, por lo tanto, un mismo usuario puede pertenecer a varios grupos. De esta forma se podrán compartir contenidos en un grupo concreto que no sean interesantes para los miembros de otro grupo, y viceversa.

A pesar de los mecanismos más o menos complejos que se esconden detrás del sistema para su organización y gestión, se puede resumir la funcionalidad de un *peer* en la capacidad de búsqueda de contenidos dentro de la red *P2P*, la elección de los más interesantes desde el punto de vista personal, y la evaluación de los mismos. Los procesos de búsqueda y actualización de los valores de reputación de contenidos y en *peers* serán descritos en los apartados siguientes.

Al igual que en el caso de la plataforma utilizada para la red *P2P*, el sistema de gestión de la reputación implementado en este proyecto ha tomado como referencia las líneas generales recogidas en la especificación del sistema *Poblano* [26]. Este sistema, *Poblano*, define las estructuras básicas para la gestión de la confianza, así como su cálculo, de forma completamente distribuida. A pesar de esto, se han tenido que modificar aspectos de la búsqueda y la actualización de contenidos para poder realizar una plena integración con la plataforma *JXTA*, tecnología a la que está orientado.

Así, se han modificado ciertos rasgos de la representación de datos del sistema original para integrarlo con el modelo de almacenamiento y gestión de datos de nuestro sistema. Por lo tanto, las estructuras de datos básicas de *Poblano* han tenido que ser modificadas para aumentar su capacidad de representación de la información, y también se han debido definir nuevas estructuras, como los mensajes, a los que la especificación inicial no hace referencia

A lo largo de este capítulo, y en el correspondiente a la implementación del sistema, se empleará el concepto de reputación, tanto en contenido como en *peer*, en lugar de confianza, como se denomina en la especificación de *Poblano*. Se recurre al concepto de reputación, en lugar de confianza, debido a que los valores calculados y compartidos en la red, se combinarán para ofrecer un valor único que defina la calidad de un *peer* como anotador, por lo que se puede considerar como un valor de reputación. Por otro lado, pueden aparecer referencias al término “confianza” o “*confidence*” debidas a que se ha tratado de mantener la nomenclatura original de *Poblano*.

3.1. Especificación de requisitos del sistema

3.1.1. Requisitos funcionales

En esta sección se hará una enumeración de los requisitos funcionales indispensables para el correcto funcionamiento del sistema. Los requisitos funcionales son aquellos que definen el comportamiento interno del software tales como cálculos, detalles técnicos, manipulación de datos y otras características específicas. Estos requisitos dan forma a lo que debe cumplir el sistema, y en nuestro caso son los siguientes:

- **Especificación *Poblano*.** El sistema deberá tomar como base de sistema de gestión de la reputación la especificación de *Poblano*, ya que es un buen punto de partida para la definición de un sistema orientado a redes *P2P*.
- **Estructuras *XML*.** El sistema deberá ser capaz de gestionar el repositorio de datos de reputación, así como los mensajes intercambiados entre los nodos, que estarán basados en *XML*.
- **Red *JXTA*.** El sistema deberá ser capaz de implementar los protocolos necesarios para integrarse en la red *P2P* definida por la plataforma *JXTA*.
- **Comunicación con otros *peers*.** El sistema deberá proporcionar los mecanismos necesarios para implementar los protocolos de petición-respuesta que se utiliza en el sistema de información para la obtención, distribución e intercambio de los valores de reputación y las tablas en las que se basa el mismo.
- **Gestión de mensajes.** El sistema deberá ser capaz de crear e interpretar, enviar y recibir mensajes para la distribución y el intercambio de valores de reputación.
- **Cálculo de valores de reputación.** El sistema deberá estar dotado de la capacidad para crear y actualizar los valores de reputación en base a las calificaciones generadas por el usuario, las recibidas como *feedback* y los valores de reputación difundidos por otros *peers*.
- **Almacenamiento de datos.** El sistema deberá poder realizar un almacenaje periódico de todos los datos contenidos en las tablas de confianza, para conseguir salvaguardar la información en caso de una hipotética caída del sistema.
- **Tiempo de guardado.** El sistema podrá establecer el intervalo de tiempo que transcurre entre almacenados en disco de las tablas de información, para garantizar la persistencia de los datos y su actualización periódica.
- **Directorio de usuario.** El sistema deberá poder configurar el directorio personal en el que se almacenará toda la información de reputación en cualquier momento de la ejecución del mismo.

- **Tiempo de espera.** El sistema deberá ser capaz de configurar el tiempo de espera que el *peer* dejará de margen para recibir respuestas una vez enviada la petición. Este tiempo puede ser más corto para favorecer el rendimiento de la aplicación, en detrimento del número de respuestas recibidas; o puede ser mayor, lo que implica que pueden ser recibidas más respuestas.
- ***TTL*.** El sistema deberá poder establecer el tiempo de vida de los mensajes generados, esto es, el número de saltos que el mensaje podrá dar entre nodos hasta que sea descartado. Este mecanismo permite llegar a nodos más alejados para obtener información de ellos.
- **Umbral de cooperación.** El sistema deberá poder establecer o cambiar en cualquier momento el umbral de cooperación aceptable para solicitar o utilizar la información recibida de un nodo remoto.
- **Elección de un *peer*.** El sistema deberá contar con un mecanismo que permita elegir a un *peer* dentro de una lista de posibles candidatos, para solicitar información de reputación, en base al umbral de cooperación.
- **Gestión de la configuración.** El sistema deberá contar con un fichero de configuración de las propiedades que será leído al inicio de la ejecución del mismo. En este fichero se podrá configurar el tiempo de guardado, el tiempo de espera para respuestas, el umbral de cooperación, el *TTL* de los mensajes y el directorio de usuario.
- **Interfaz de aplicación.** El sistema deberá proporcionar las interfaces necesarias para ser utilizado desde cualquier aplicación externa, permitiéndola acceder a toda su funcionalidad.
- **Biblioteca.** El sistema deberá proporcionar una biblioteca para que cualquier aplicación que la incluya pueda acceder a toda la funcionalidad del sistema.
- **Interfaz de usuario.** El sistema deberá proporcionar una interfaz de usuario que permita la gestión del mismo, así como realizar pruebas por parte del administrador.
- **Gestión de *logs*.** El sistema deberá contar con un mecanismo de gestión de *logs* que permita conocer en todo momento el uso dado al mismo, así como registrar hipotéticos errores de ejecución.

3.1.2. Requisitos no funcionales

Para completar la definición de un sistema o aplicación, los requisitos no funcionales complementan a los funcionales, describiendo criterios que son necesarios para el funcionamiento del sistema, pero que no tienen que ver con el comportamiento específico del mismo. También son conocidos como parámetros de calidad, y para nuestro sistema de gestión de la reputación son:

- **Espacio en disco.** El sistema deberá contar con espacio suficiente en disco para almacenar toda la información de las tablas de confianza así como las librerías necesarias para la ejecución del programa.
- **Permisos de lectura/escritura.** El sistema deberá contar con los permisos de lectura y escritura sobre los directorios establecidos para cada usuario del mismo.
- **Maquina Virtual de Java.** El equipo en el que se ejecute el sistema deberá contar con una *JVM* para la ejecución del mismo.
- **Portabilidad.** El sistema será capaz de ejecutarse en cualquier equipo, sobre cualquier sistema operativo.
- **Interoperabilidad.** El sistema será capaz de interactuar con otros *peers* que se ejecuten en distintas plataformas y que estén escritos en diferentes lenguajes de programación. Para cumplir con este objetivo se emplean la plataforma *JXTA* y la especificación de *Poblano* como ejes vertebradores del mismo.
- **Conectividad.** El equipo en el que se ejecute el sistema deberá contar con total conectividad a la red para su correcto funcionamiento, permitiendo así la interconexión con otros nodos de la red.
- **Documentación.** El sistema estará bien documentado. El código fuente contará con comentarios suficientes para el entendimiento del mismo, y el proyecto se respaldará con una memoria técnica explicativa.
- **Coste.** El desarrollo del sistema deberá suponer unos costes lo suficientemente ajustados para que su precio de venta sea competitivo en el mercado.

3.2. Arquitectura del sistema

Para poder entender el funcionamiento del sistema sobre el que trata esta memoria se debe dar una primera visión general de su mecánica. En este apartado se intentará abordar ésta cuestión presentando una serie de puntos que permitirán conocer las bases del sistema, de tal forma que cualquier persona que los consulte pueda comprender cómo funciona.

No pretende ser ésta una sección que explique todos los detalles de la implementación llevada a cabo, pero sí está pensado para que cualquier lector pueda tener las nociones suficientes sobre la estructura global del sistema y su comportamiento. Con el apoyo del capítulo dedicado al desarrollo del sistema, en el que se podrá ver la estructuración en módulos, los interfaces y las estructuras de datos fundamentales, cualquier desarrollador de aplicaciones estará en posición de realizar un sistema similar a éste u otro que permita una total interoperabilidad con el mismo.

3.2.1. Estructura general

Antes de comenzar con explicaciones más concretas del sistema, se debe mostrar una visión general del mismo en la que se puedan ver los bloques básicos en los que está dividido. Esto ayudará en gran medida al entendimiento de su desarrollo, que se podrá consultar en el capítulo que trata la implementación del sistema.

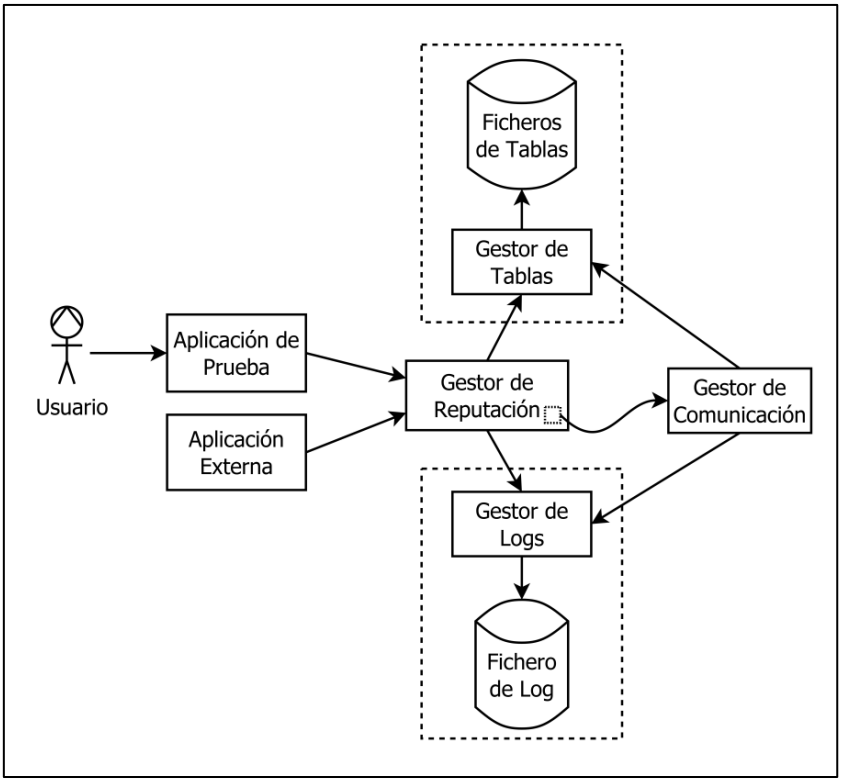


Figura 14: Arquitectura en bloques del sistema

Tal y como se puede ver en la figura 14, el sistema gira en torno a un componente central que será el propio bloque gestor de la reputación. Este módulo será el encargado de proporcionar las interfaces necesarias para que cualquier aplicación pueda acceder a toda la funcionalidad del mismo. Como se puede ver en dicha figura, un usuario podrá hacer uso de este sistema a través de una aplicación de prueba.

Este bloque gestor de reputación, como se puede comprobar en la figura antes mencionada, será el intermediario entre las aplicaciones que hagan uso del sistema y los datos propiamente dichos que manejará, conectándose con otros dos módulos que serán los encargados de gestionar las tablas de confianza y los ficheros de *log*.

El primero de estos bloques es el encargado de la gestión de las tablas, que tendrá una doble función: por una parte su cometido será el de acceder al repositorio de datos correspondientes a los valores de reputación, tanto en contenidos (es decir, anotaciones) como en *peers*, que estará almacenado en disco; por otro lado también se encargará de mantener en memoria toda esta información a lo largo de la ejecución del sistema para agilizar el proceso de acceso a los datos.

El segundo bloque al que está conectado el núcleo gestor de la reputación será el encargado de manejar los *logs* del sistema. Este módulo llevará todo el peso de la lectura y escritura en disco de la información susceptible de almacenarse como entrada de *log*. Entre esta información se pueden encontrar las trazas de todas las comunicaciones llevadas a cabo por el *peer*.

El último de los módulos que se pueden encontrar en el sistema es el encargado de la gestión de las comunicaciones con usuarios o aplicaciones remotas. Este módulo de comunicación se puede considerar un bloque interno del gestor de la reputación que, independientemente de este, estará conectado con los módulos gestores de tablas y *logs*.

Una vez vista la arquitectura real del sistema se puede pasar a ver aspectos más concretos de su funcionamiento, como el manejo de la información de la reputación por parte del sistema y el intercambio de mensajes, que se verán en los siguientes apartados.

3.2.2. Búsqueda de contenidos

Como se ha mencionado anteriormente, el sistema se basa en la división de la información por grupos de intereses y, a falta de un conjunto de anotaciones locales sobre el que poder realizar búsquedas, se debe recurrir a *peers* remotos que nos faciliten dicha información. Cuando un *peer* acaba de acceder a un determinado grupo de la red, es probable que no conozca a nadie de dicho grupo a quien enviar peticiones en caso de necesidad. En este caso el sistema deberá dotar a cada usuario de la capacidad de enviar y recibir tablas completas de confianza en *peers*. De esta forma, un determinado *peer* que se conozca dentro un grupo *"i"*, podrá facilitar al usuario local toda su tabla de confianza en *peers* para el grupo *"j"* al que acaba de acceder, proporcionando así un conjunto de fuentes de contenidos más amplia que el que se tenía en principio.

Tal y como se verá en detalle en el apartado dedicado a las estructuras fundamentales del sistema, cada *peer* posee tablas de confianza en anotaciones semánticas y tablas de confianza en *peers* diferentes para cada grupo en el que participa, y que serán la base para que cualquier búsqueda se realice correctamente.

Cada vez que un usuario realiza una búsqueda deberá seguir los siguientes pasos principalmente:

- Buscará la palabra clave en sus propias tablas de confianza en anotaciones. Si hay algún contenido local asociado con dicha palabra clave entonces la búsqueda habrá sido exitosa. En caso contrario se deberá pasar al siguiente punto.
- Buscará la palabra clave en sus propias tablas de confianza en *peers*. Si existe en ellas algún *peer* que podamos asociar con dicha palabra clave enviaremos una petición a cada uno de estos *peers* encontrados. Estos *peers*, a su vez, ejecutarán el primero y el segundo de estos pasos en sus propias tablas. Si alguno de estos *peers* encuentra un contenido referente a

dicha palabra clave enviará una respuesta al primer *peer* y finalizará la búsqueda. En caso contrario se pasará al siguiente punto.

- Buscará la palabra clave en los anuncios que envían otros *peers* por la red, que indicará que esos *peers* poseen anotaciones que concuerdan con la búsqueda. Si se encuentra alguno, se le enviará una petición como ocurría en el punto anterior, y dicho peer realizará una búsqueda en sus tablas locales tal y como se especifica en los dos primeros pasos.

Si todo esto falla se considerará que la búsqueda no ha sido exitosa, y por lo tanto el *peer* no encontrará ningún contenido asociado a la palabra clave que estaba buscando.

Este proceso se puede afinar aún más, si consideramos que el sistema se basa en tablas diferentes para distintos grupos de intereses. En este caso, cada uno de los pasos de la búsqueda se hará en primer lugar en las tablas correspondientes al grupo de intereses en el que se realiza la búsqueda y, si no se encuentra ningún resultado, se recurrirá a consultar las tablas de confianza de otros grupos, como se puede ver en la figura 15.

Por otro lado se le da al usuario la opción de forzar la búsqueda para que sea remota, esto es, ignorando los posibles resultados almacenados en las tablas de confianza en contenido locales y buscando directamente información en *peers* remotos. Si se da este caso, los componentes sombreados del diagrama siguiente serán ignorados, ya que representan la búsqueda en las tablas locales de anotaciones.

Como dato adicional, es importante resaltar que las anotaciones semánticas son los contenidos, que en cualquier otro sistema podrían ser cualquier recurso compartido en la red, desde un fichero de música hasta un archivo de texto. Por tanto se utilizarán ambos términos de forma indistinta.

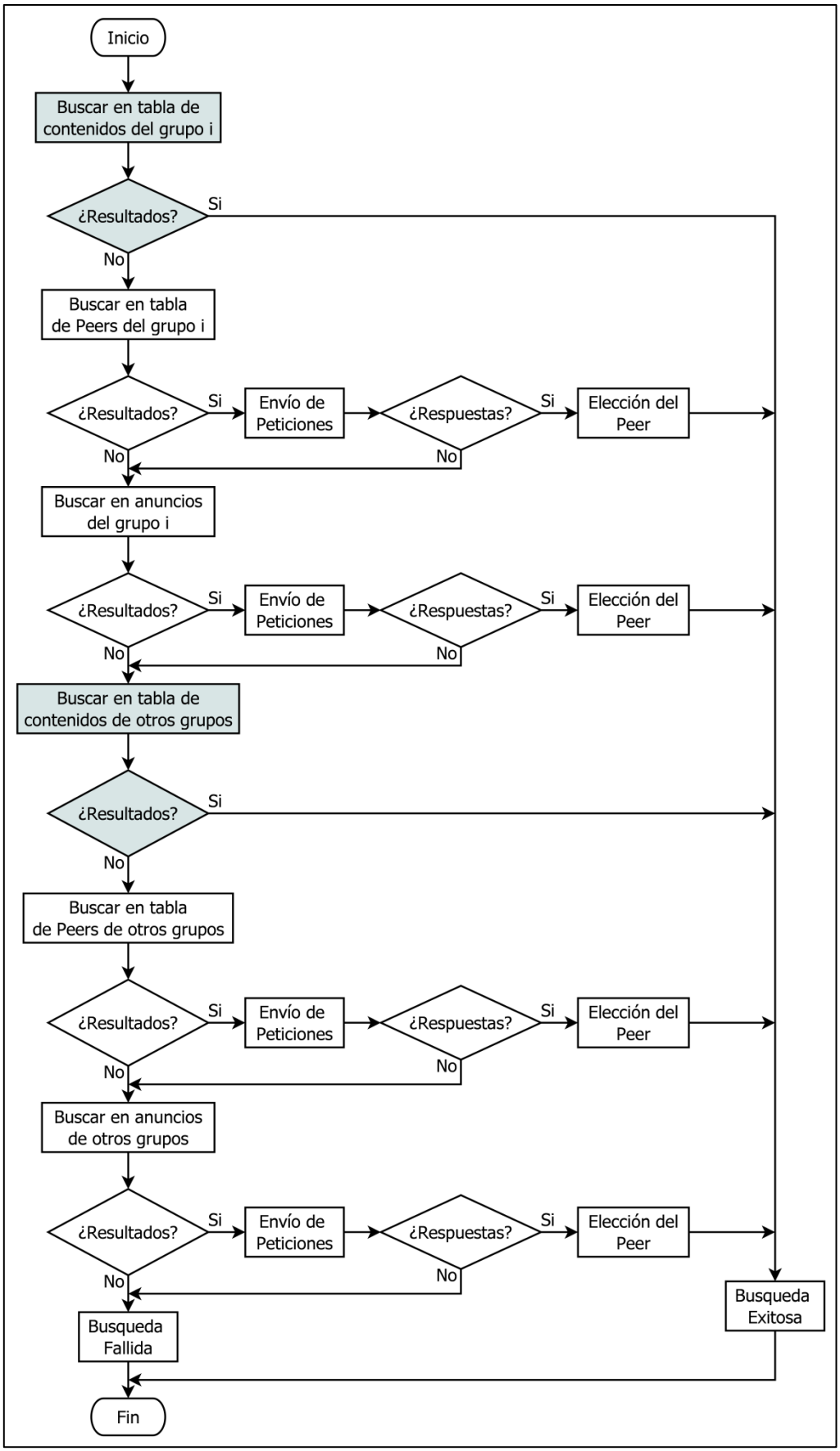


Figura 15: Diagrama de flujo de una búsqueda

Como se puede ver en el diagrama de la figura 15, en cualquier punto de la búsqueda se pueden realizar peticiones de información a *peers* remotos. Desde el punto de vista de uno de estos *peers* que recibe una petición de búsqueda, el proceso de consulta en sus tablas locales será el mismo, es decir, primero buscará en las tablas de confianza en contenido locales y después pasará a buscar en sus tablas de confianza en *peers* para reenviar la petición a sus conocidos.

Para controlar la inundación de mensajes de petición de información en la red se ha establecido el uso de un tiempo de vida para cada uno de ellos. En cada salto que dé el mensaje, es decir, cada *peer* que reciba el mensaje, si necesita reenviarlo de nuevo, deberá hacerlo disminuyendo en una unidad este tiempo de vida. Si un usuario determinado recibe un mensaje de petición cuyo tiempo de vida es igual a cero, este buscará la información solicitada en sus tablas locales, pero no reenviará el mensaje.

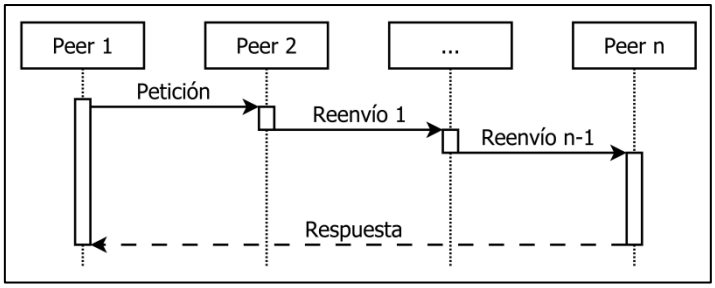


Figura 16: Diagrama de secuencia de un reenvío de mensajes

En la situación particular que supone el reenvío de mensajes, como muestra la figura 16, se puede dar el caso de que un mensaje haya pasado por varios *peers* intermediarios hasta dar con uno que cuente con la información solicitada. Este último *peer* deberá enviar un mensaje de respuesta directamente al *peer* que originó la petición, en lugar de enviar la información de nuevo hacia atrás en la cadena de *peers* por la que se recibió. El motivo de este comportamiento se debe exclusivamente la optimización del rendimiento del sistema, evitando así la inundación de la red con mensajes reenviados y disminuyendo la carga de trabajo de los *peers* intermedios.

3.2.3. Actualización de la reputación

Aunque el cálculo concreto de los nuevos valores de reputación será discutido más adelante en la sección de desarrollos algorítmicos, en este apartado se verá el proceso de actualización de los valores de reputación. Para conocer más detalles de estos cálculos se puede consultar dicha sección.

Este proceso comienza cuando un *peer*, ya sea un usuario o una aplicación externa, accede a un contenido. Cada vez que esto ocurre, el valor de popularidad de dicho contenido se incrementará en uno automáticamente. Por otra parte, el valor de confianza en dicho contenido se actualizará gracias a la puntuación que el usuario le otorgue. Esta puntuación podrá ir desde -1 si el contenido no se ajusta a lo buscado hasta 4, que será la mejor calificación para un contenido.

Además de la puntuación del propio usuario, para la actualización del valor de reputación también se tendrá en cuenta la información propagada por otros *peers* conocidos. Esta información será obtenida bajo demanda del *peer* local, que enviará mensajes de petición para obtener dichos valores difundidos. La respuesta de cada uno de los *peers* será el valor de reputación para dicho contenido que tenga en su tabla local. En última instancia, para calcular el nuevo valor de reputación también se tendrá en cuenta el antiguo valor almacenado en las tablas locales, en caso de que existiera.

Si dicho contenido se ha obtenido de un *peer* remoto, el usuario local añadirá una entrada a su tabla local de confianza en contenidos, pero ahí no acaba el proceso de actualización. El conjunto de todas esas puntuaciones que el *peer* local ha dado a los contenidos obtenidos de ese usuario remoto serán utilizados para generar o actualizar un valor de reputación en *peer* con respecto a una palabra clave concreta. Este proceso puede verse en la figura 17.

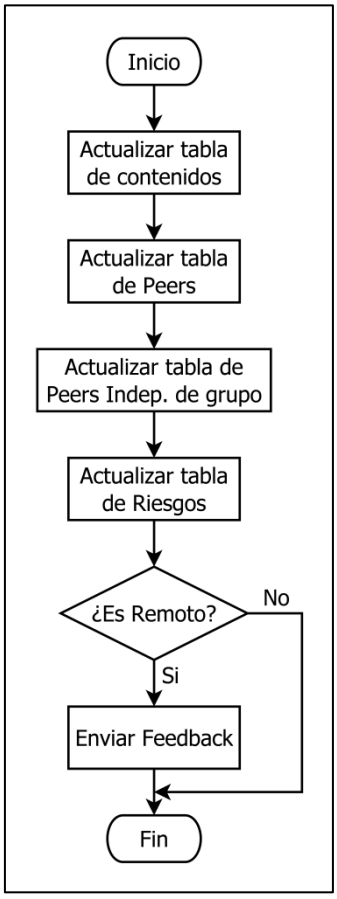


Figura 17: Diagrama de flujo actualización de reputación

Desde el punto de vista del *peer* que facilitó dicho contenido, éste recibirá un *feedback* de la puntuación que el primer *peer* le asignó al mismo. De esta forma, dicho *peer* actualizará su valor de reputación del contenido y también el valor de reputación del *peer* que le envió el *feedback*. Vemos que en este caso también se actualizan todas las tablas, desde la de confianza en contenidos hasta la de riesgos.

3.2.4. Configuración del sistema

Antes de comenzar la ejecución, e incluso durante la misma, el sistema tendrá la capacidad de configurar ciertos parámetros que permitirán ajustar su comportamiento. Para ello, el sistema cuenta con un fichero de configuración que se podrá modificar al gusto del usuario.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <entry key="ttl">1</entry>
    <entry key="userPath">.</entry>
    <entry key="saveTime">10000</entry>
    <entry key="waitTime">1500</entry>
    <entry key="pBest">0.6</entry>
</properties>
```

Figura 18: Detalle del fichero de configuración

Este fichero posee una estructura *XML* muy sencilla, tal y como se puede ver en la figura 18, que permitirá configurar los siguientes parámetros:

- **Tiempo de vida:** Permite establecer el número de saltos máximo que puede realizar un mismo mensaje dentro de la red antes de ser descartado. Esto permitirá que, si se necesita reenviar un mensaje de petición, este no se propague indefinidamente por la red.
- **Directorio de Usuario:** Permite definir, mediante una cadena de caracteres, la ruta completa o relativa al directorio personal que el usuario utilizará para almacenar toda la información relativa al sistema de gestión de la reputación, desde las tablas de confianza, hasta los ficheros de *log*.
- **Tiempo de guardado:** El sistema realizará un almacenado periódico de las tablas de confianza para evitar posibles pérdidas de datos y garantizar la persistencia de los mismos en disco. Mediante este parámetro se determinara el tiempo, en milisegundos, entre guardados consecutivos.
- **Tiempo de espera:** Cada vez que un nodo realiza una petición a la red, no se puede determinar a priori cuánto tardará en recibir respuestas, ya que el rendimiento de unos nodos es distinto al de otros. Por este motivo, se necesita un parámetro que determine el tiempo máximo, en milisegundos, que el *peer* esperará hasta comenzar el procesado de los datos recibidos.
- **Probabilidad de elección:** Este parámetro es muy importante dentro del sistema de gestión de la reputación, ya que trata de evitar posibles sobrecargas de los mejores nodos de la red. En realidad, este valor es un número decimal entre 0 y 1, que determinará la probabilidad de solicitar información al *peer* que tiene mejor reputación de entre los posibles candidatos que cuentan con ella.

3.3. Estructuras de datos fundamentales

Todos los mecanismos que constituyen el funcionamiento del sistema necesitan de la existencia de una serie de estructuras básicas para el manejo de los datos, y también para su intercambio. La utilización de un sistema unificado de estructuras por todos los *peers*, o usuarios del sistema de reputación, proporciona los fundamentos de la portabilidad del mismo, haciéndolo independiente de la plataforma en la que se esté ejecutando.

Por un lado están todos aquellos modelos de datos que se centran en el almacenamiento de la información de forma local. Estos permitirán que todos los usuarios almacenen los datos de la misma forma, y por lo tanto que se pueda migrar de un sistema a otro sin necesidad de efectuar cambios en el código que los gestiona.

En el otro extremo están aquellas estructuras que se emplean durante la intercomunicación de los *peers*. En este caso, el modelo único que se utiliza permitirá que cualquier usuario que acceda al sistema de gestión de la reputación no tenga problemas de comunicación sea cual sea la plataforma desde la que se conecte y el dispositivo que se esté utilizando.

En los apartados siguientes se mostrará la especificación de cada una de estas estructuras, explicando sus variantes y sus elementos constituyentes, de forma que cubran las diferentes posibilidades para cada una de ellas y sea más fácil la comprensión de su significado.

3.3.1. Tipos de Mensaje

En los apartados anteriores se ha hecho referencia en varios puntos al intercambio de información entre usuarios. Todo este trasvase de información se realiza mediante un protocolo de petición-respuesta que permitirá el flujo de toda la información que los *peers* pudieran necesitar para el normal funcionamiento del sistema.

El formato específico de cada uno de los mensajes del sistema se puede consultar en la sección correspondiente a las estructuras fundamentales, por lo que aquí haremos un breve repaso de todos ellos comentando su utilidad.

Como es de esperar, para cada tipo de mensaje existe una petición y su correspondiente respuesta. Los posibles mensajes en nuestro sistema son los siguientes:

- **Mensaje de palabra clave (KEYW):** Este mensaje se origina durante la búsqueda de un contenido. En la petición, un *peer* generará un mensaje en el que se especificará tanto un grupo de intereses como una palabra clave para que el *peer* remoto le devuelva la correspondiente respuesta. Esta respuesta estará formada por todos los contenidos que se ajusten al mismo grupo y palabra clave especificados en la petición.

- **Mensaje de contenido (CONT):** Este mensaje se origina durante la evaluación de un contenido, para solicitar la propagación de un valor de reputación del contenido. En este caso la petición llevará toda la información del contenido que está siendo evaluado, y la respuesta contendrá el valor de reputación y la popularidad que el *peer* remoto tiene almacenado para el mismo.
- **Mensaje de calificación (RATE):** Este mensaje se origina tras la evaluación de un contenido en local, y sirve para enviar el *feedback* a un *peer* remoto. El mensaje de petición contendrá toda la información del contenido, así como el valor de reputación actualizado para el mismo. El mensaje de respuesta será un acuse de recibo de la petición.
- **Mensaje de tabla (TABL):** Este mensaje se origina durante la actualización de una tabla de confianza en contenidos por parte de un *peer*. El mensaje de petición indicará el grupo de intereses cuya tabla se desea actualizar. El mensaje de repuesta del *peer* remoto contendrá toda la tabla de confianza en *peers* para dicho grupo.

Este apartado está dirigido a explicar los mensajes que los *peers* generarán a lo largo de su vida, mecanismo básico necesario para el intercambio de información entre los usuarios del sistema de gestión de la reputación. Cada vez que un *peer* desee obtener cualquier tipo de información de la red, tendrá que adaptarse a las directrices de un mecanismo de petición-respuesta definido previamente, para garantizar que todos ellos sean capaces de entenderse y llevar a cabo la comunicación correctamente.

3.3.1.1. Campos comunes

Todos los mensajes tendrán una parte fundamental que se corresponde con el tipo de información que se quiere intercambiar y que será independiente para cada caso. Esta información será la que realmente importa en cada mensaje, y puede ser considerada como la carga útil del mismo. A pesar de esto, todos los mensajes, tanto de petición como de respuesta contendrán unos determinados campos comunes, que serán necesarios tanto para el encaminamiento del mensaje como para tareas del procesado de los mismos. Esta parte común contiene los siguientes campos:

- **`Type:`** Es el tipo de mensaje con el que se está tratando. Será un valor numérico que indicará si el mensaje es una petición (1-*RRQST*), una respuesta (2-*RRPLY*), o un mensaje de alerta (3-*RALRT*).
- **`Subtype:`** Es el subtipo del mensaje con el que se trabaja. Será un número entero que podrá tomar los valores: 1 (*KEYW*), en caso de que se trate de la búsqueda de contenidos según su palabra clave; 2 (*CONT*) en caso de que se trate de la búsqueda del valor de reputación de un contenido en concreto; 3 (*RATE*) si se trata de un mensaje que forma parte de la evaluación de un contenido; o 4 (*TABL*) en caso de que se trata de un mensaje para el intercambio de tablas de confianza en *peer*.

- **SenderID:** Es el identificador *JXTA* único correspondiente al *peer* que genera el mensaje.
- **SenderName:** Es el nombre del *peer* que genera el mensaje.
- **RecipientID:** Es el identificador *JXTA* único correspondiente al *peer* que recibe el mensaje.
- **MessageID:** Es el identificador *JXTA* único utilizado para identificar el mensaje, y que servirá para emparejar peticiones con sus correspondientes respuestas.
- **TTL:** Es un valor numérico entero que representa el tiempo de vida del mensaje. Esto determina el número de saltos que dicho mensaje podrá dar a lo largo de la red antes de ser descartado o, dicho de otro modo, el número de veces que se podrá reenviar.

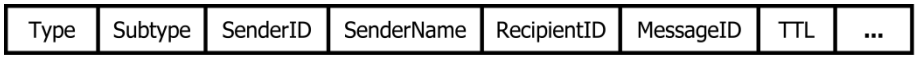


Figura 19: Formato común de los mensajes

Como se ha comentado anteriormente, estos campos del mensaje forman la base para el tratamiento del mensaje, así como para conseguir que dicho mensaje llegue al destinatario correcto. A partir de ahora continuaremos con la explicación de cada uno de los tipos de mensaje, con sus particularidades.

3.3.1.2. Peticiones

En primer lugar haremos un repaso de los mensajes de petición, que serán el inicio de toda comunicación dentro del sistema de gestión de la reputación. Cada uno de estos mensajes se podrá generar en cualquier momento a petición del usuario, gracias a los métodos desarrollados en el apartado correspondiente.

KEYW

Este mensaje se genera cada vez que un *peer* desea consultar a un nodo remoto por todos los contenidos que posee dentro de un grupo determinado y para una palabra clave determinada. Se puede observar que, por lo general, en cada búsqueda el *peer* enviará peticiones a múltiples usuarios de la red, y por lo tanto se generará un mensaje diferente por cada destinatario. Los campos particulares de este tipo de mensajes son:

- **Keyword:** Es la palabra clave para la que el usuario está realizando la búsqueda.
- **PeerGroupID:** Es el identificador *JXTA* único en el que el usuario realiza la búsqueda.
- **PeerGroupName:** Es el nombre del grupo objeto de la búsqueda.

Figura 20: Formato mensaje de petición KEYW

CONT

Los usuarios que generan este mensaje pretender consultar los valores de reputación que los *peers* destinatarios poseen en sus tablas locales para un contenido determinado. Esto ocurre cuando, tras la búsqueda de un contenido y su correspondiente acceso, el *peer* procede a la evaluación del mismo. En dicha evaluación se deberá incluir un término correspondiente a la opinión propagada por otros *peers*, y este es el mensaje que se encarga de pedirlos. Los campos propios de este tipo de petición son:

- **`Keyword:`** Es la palabra clave a la que hace referencia el contenido que se está consultando.
- **`PeerGroupID:`** Es el identificador *JXTA* único del grupo en el que se realiza la búsqueda.
- **`PeerGroupName:`** Es el nombre del grupo en el que se realiza la consulta.
- **`ContentID:`** Es el identificador *JXTA* único del contenido que se quiere consultar.
- **`ContentName:`** Es el nombre de dicho contenido.

Figura 21: Formato mensaje petición CONT

RATE

Este tipo de petición se utilizará para enviar a un *peer* remoto el nuevo valor de reputación que se le ha asignado a un contenido que él mismo nos envió. Esto puede verse como el envío de un *feedback* a dicho usuario, para que esté al tanto de las opiniones que tienen otros *peers* de la red sobre ese contenido. Por tanto, los campos particulares de este tipo de petición serán:

- **`Keyword:`** Es la palabra clave a la que hace referencia el contenido que se ha evaluado.
- **`PeerGroupID:`** Es el identificador *JXTA* único del grupo en el que se realizó la búsqueda.
- **`PeerGroupName:`** Es el nombre del grupo en el que se realizó la consulta.
- **`ContentID:`** Es el identificador *JXTA* único del contenido del que se quiere enviar el *feedback*.
- **`ContentName:`** Es el nombre de dicho contenido.

- **`Relevance`:** Es el valor de reputación.

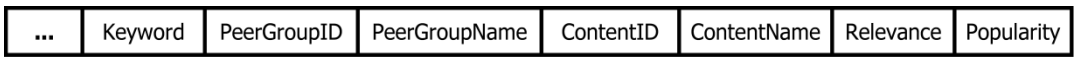


Figura 22: Formato mensaje petición RATE

TABL

Este tipo de petición se generará cuando un usuario desee actualizar su tabla local de confianza en *peers*, principalmente con el objetivo de conocer la identidad de nuevos usuarios a los que poder consultar contenidos, ampliando así las posibilidades de encontrar resultados que se ajusten a dichas búsquedas. Esta petición se enviará a uno de los *peers* remotos que el usuario del sistema local tenga almacenado previamente en sus tablas, el cual enviará los identificadores de los nodos que conozca para así ampliar la tabla local de confianza en *peers*. Los campos particulares de este mensaje de petición son:

- **`PeerGroupID`:** Es el identificador *JXTA* único del grupo para el que se desea actualizar la tabla de confianza en *peers*.
- **`PeerGroupName`:** Es el nombre del grupo para el que va dirigida la petición.

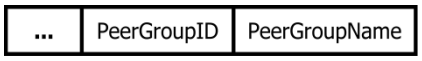


Figura 23: Formato mensaje petición TABL

Se acaba de realizar un repaso por todos los mensajes de petición susceptibles de ser creados a lo largo de la vida del sistema de gestión de la reputación. Como se ha podido ver, cada petición recibida por un *peer* hará que se genere automáticamente un mensaje de respuesta que será el encargado de hacer llegar al usuario que lo generó la información solicitada.

3.3.1.3. Respuestas

Cada subtipo de mensaje de petición tendrá un subtipo de mensaje de respuesta concreto, por lo tanto, existirán cuatro tipos de mensajes de respuesta. Se debe recordar en este punto que, al igual que los mensajes de petición, las respuestas también contarán con los campos comunes a todos los mensajes que vimos al comienzo de esta sección. A partir de aquí, se describirán los campos propios de cada mensaje de respuesta según su subtipo:

KEYW

Cuando un *peer* recibe una petición de tipo *KEYW* es porque un usuario remoto desea conocer los contenidos locales almacenados en las tablas para una palabra clave concreta y dentro de un grupo determinado. Por lo tanto, esto generará una búsqueda en las tablas locales cuyos resultados se añadirán al mensaje. El mensaje de respuesta, por tanto, deberá contar con los siguientes campos:

- **`Keyword:`** Palabra clave para la que el *peer* remoto desea conocer los contenidos almacenados en nuestras tablas locales.
- **`PeerGroupID:`** Identificador *JXTA* único en el que el *peer* remoto desea conocer los contenidos buscados, y por lo tanto, en el que se habrá realizado la búsqueda.
- **`PeerGroupName:`** Nombre del grupo en el que el *peer* remoto desea conocer los contenidos buscados, y por lo tanto, en el que habremos realizado la búsqueda.
- **`Resultados:`** El mensaje podrá contener uno o más campos correspondientes a los resultados arrojados por la búsqueda. Cada uno de estos campos se compondrá de una cadena de caracteres en el que estarán concatenados identificador y nombre de un contenido separados por el carácter ";". La motivación de esta solución se basa en que el *peer* al que se envía la respuesta puede no conocer todos los nombres de los contenidos que se han encontrado en las tablas locales, por lo que habrá que hacérselos llegar de alguna forma para que los pueda visualizar de una forma más amigable.

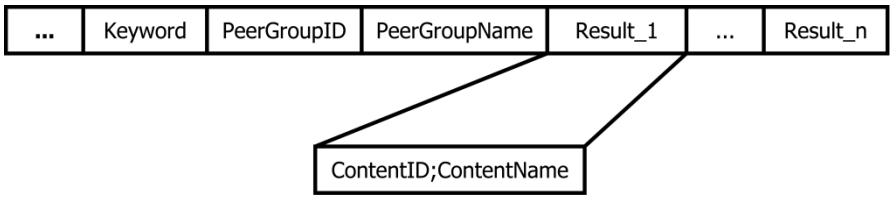


Figura 24: Formato mensaje respuesta KEYW

CONT

Este mensaje de respuesta se genera cuando el sistema recibe un mensaje de consulta de un contenido específico. Esto suele darse cuando un *peer* remoto desea recopilar toda la información propagada por el resto de usuarios de la red con el objeto de llevar a cabo la evaluación de un contenido. Esto, desde el punto de vista local, va a suponer una búsqueda de un valor de reputación de un contenido concreto, que estará almacenado en las tablas. Los campos particulares de este mensaje son:

- **`Keyword:`** Palabra clave a la que hace referencia el contenido cuya reputación desea consultar el *peer* remoto.
- **`PeerGroupID:`** Identificador *JXTA* único en el que el *peer* remoto está realizando la evaluación de dicho contenido.
- **`PeerGroupName:`** Nombre del grupo en cuestión.
- **`ContentID:`** Identificador *JXTA* único del contenido que el *peer* remoto quiere consultar.
- **`ContentName:`** Nombre del contenido correspondiente.

- **`Relevance`:** Valor de reputación de dicho contenido almacenado en las tablas locales del sistema.
- **`Popularity`:** Valor de popularidad que tiene dicho contenido en las tablas locales del sistema.

...	Keyword	PeerGroupID	PeerGroupName	ContentID	ContentName	Relevance	Popularity

Figura 25: Formato mensaje respuesta CONT

RATE

Esta respuesta se genera cuando un *peer* remoto envía a nuestro sistema el *feedback* correspondiente a la puntuación de un contenido. La reputación de dicho contenido habrá sido actualizado en las tablas de dicho usuario de acuerdo con la puntuación asignada por el mismo, y teniendo en cuenta la información propagada por otros *peers*. Por este motivo el sistema tendrá que actualizar la tabla local de confianza en contenidos para recoger esta nueva evaluación. Una vez lo haya hecho se enviará esta respuesta para confirmar al *peer* remoto que se ha recibido su petición. Los campos de la respuesta son:

- **`Keyword`:** Palabra clave a la que hace referencia el contenido del que el sistema ha recibido el *feedback*.
- **`PeerGroupID`:** Identificador *JXTA* único para el que el *peer* remoto evaluó el contenido.
- **`PeerGroupName`:** Nombre de dicho grupo.
- **`ContentID`:** Identificador *JXTA* único del contenido evaluado por el *peer* remoto.
- **`ContentName`:** Nombre que corresponde a este contenido.
- **`Updated`:** Campo que indica si el valor de reputación del contenido ha sido actualizado correctamente en base al *feedback* recibido.

...	Keyword	PeerGroupID	PeerGroupName	ContentID	ContentName	Updated

Figura 26: Formato mensaje respuesta RATE

TABL

Las peticiones de este subtipo se producen cuando un usuario desea ampliar el conocimiento de *peers* dentro de la red gracias a las tablas locales de confianza en *peers*, en previsión de que nuestro sistema habrá tenido contacto con nodos que dicho usuario desconoce. Cuando uno de estos mensajes llega al sistema, se producirá una búsqueda sobre la tabla de confianza en *peers* local para el grupo concreto en el que el usuario remoto desea ampliar sus conocimientos. Los campos que incluirá el mensaje de respuesta correspondiente serán los siguientes:

- **`PeerGroupID:`** Identificador *JXTA* único para el que el *peer* remoto quiere ampliar sus registros de nodos.
- **`PeerGroupName:`** Nombre del grupo correspondiente.
- **`Resultados:`** Habrá uno o más campos correspondientes a los resultados encontrados en las tablas locales. Cada uno de estos campos estará compuesto por una cadena de caracteres en la que estarán el identificador del *peer* encontrado, su nombre, la palabra clave para la que tenemos su reputación, el valor de popularidad, su valor de reputación propiamente dicho, el número de contenidos recibidos de él, la suma de las reputaciones de dichos contenidos y la marca de tiempo correspondiente a la última actualización de la reputación. Todos estos campos estarán separados por el carácter ";" para facilitar su procesamiento en el *peer* remoto.

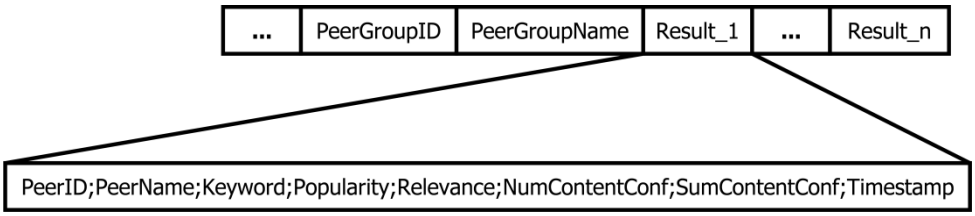


Figura 27: Formato mensaje respuesta TABL

3.3.2. Tablas de confianza.

En el apartado anterior se ha visto cómo están constituidos los mensajes que los *peers* intercambian en la red, dejando cubierto de esta forma el ámbito de comunicación entre los usuarios del sistema de reputación. En esta sección nos centraremos en la parte local del sistema, realizando una descripción detallada de todas las tablas que deben estar almacenadas en memoria (y que serán guardadas en disco entre ejecuciones) y el formato que deberán tener para que todo funcione correctamente.

Llegados a este punto se debe hacer una advertencia, ya que cada una de las tablas de confianza puede ser vista de tres formas distintas, dependiendo de la perspectiva que se utilice.

El primero de los puntos de vista es el de mayor abstracción, y puede ser considerado como el enfoque que utilizará el usuario que acceda al sistema. Mediante esta perspectiva, se pueden ver las tablas como tales, es decir, como un conjunto de datos organizados al estilo de una base de datos en el que cada entrada se corresponderá con un valor de reputación determinado. Este enfoque se ha discutido ya en el apartado correspondiente a *Poblano* dentro del capítulo del estado del arte, por lo que no incidiremos más en él.

El segundo de los puntos de vista es el que se corresponde con el almacenamiento de los datos en el disco. Debe ser tenido en cuenta por cualquier aplicación que quiera realizar un procesado de la información contenida en las tablas de confianza, si desea

acceder directamente a los ficheros. Desde esta perspectiva, las tablas de confianza son simples ficheros *XML* que deberán ser procesados para obtener todos datos correspondientes a los valores de reputación propiamente dichos, pero que son una herramienta muy poderosa para representar los datos almacenados. En este segundo nivel de abstracción nos encontramos con que las tablas se corresponden con cuatro archivos *XML*, uno para la reputación de contenidos, dos para la reputación de *peers* (uno para la dependiente de grupos y otro para la independiente), y una cuarta correspondiente a los riesgos.

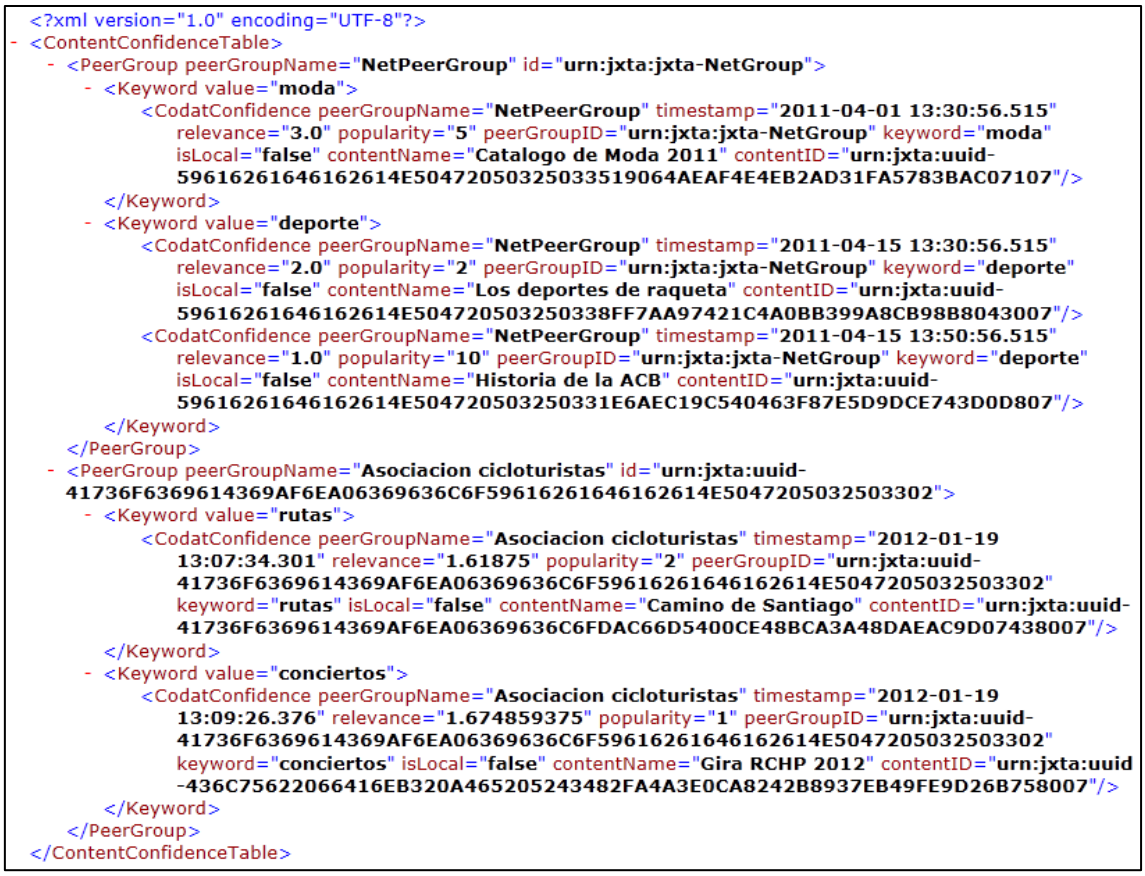
El tercer y último punto de vista será transparente para el usuario, y debe ser tenido en cuenta únicamente para los desarrolladores. Este nivel se corresponde a cómo es tratada la información de la reputación dentro del sistema, es decir, a qué tipo de datos se enfrenta internamente la aplicación a lo largo de su ejecución, pero que solamente afectará a dicho sistema, sin interferir en los dos niveles de representación anteriores, ni tampoco en la forma que tendrán los *peers* para intercambiar información. En este caso debemos decir que el modelo empleado sí es dependiente de la plataforma en la que se programará el sistema.

En nuestro caso, el lenguaje de programación utilizado es *Java*, por lo que hemos tenido que recurrir al *API* correspondiente para encontrar una estructura que pudiese representar nuestras tablas de confianza. Para ello hemos decidido utilizar una serie de tablas de *hash* (`java.util.Hashtable`) que se encargarán de organizar los datos de una forma concreta para que el sistema pueda manejarlos. Esta forma de ver las tablas ya ha sido suficientemente detallada en el apartado correspondiente a la estructura en módulos del sistema, por lo que no incidiremos de nuevo en ella.

Una vez dicho esto podemos pasar a ver en detalle cada una de las tablas desde estos puntos de vista:

- **Tabla de confianza en contenidos**

Desde el punto de vista del almacenamiento de datos, nos encontramos con que este conjunto de tablas de confianza en contenidos puede resumirse en un único fichero *XML*. En este fichero tendremos un elemento de nivel superior correspondiente a la etiqueta `<ContentConfidenceTable>`. Dentro de este elemento podrá haber uno o varios elementos de tipo `<PeerGoup>`, que se corresponderán con cada una de las tablas vistas anteriormente para cada grupo. Cada elemento `<PeerGoup>` contendrá a su vez uno o varios elementos de tipo `<Keyword>`, que se corresponderán con cada una de las tablas correspondientes a las palabras clave. Por último, cada uno de estos elementos contendrá uno o varios elementos de tipo `<CodatConfidence>` que representarán en último término el valor de reputación del contenido. Un ejemplo de este fichero sería el mostrado en la figura 28.



```
<?xml version="1.0" encoding="UTF-8"?>
- <ContentConfidenceTable>
  - <PeerGroup peerGroupName="NetPeerGroup" id="urn:jxta:jxta-NetGroup">
    - <Keyword value="moda">
        <CodatConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-01 13:30:56.515"
          relevance="3.0" popularity="5" peerGroupID="urn:jxta:jxta-NetGroup" keyword="moda"
          isLocal="false" contentName="Catalogo de Moda 2011" contentID="urn:jxta:uuid-
          59616261646162614E504720503250335190 64AEAF4E4EB2AD31FA5783BAC07107"/>
      </Keyword>
    - <Keyword value="deporte">
        <CodatConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-15 13:30:56.515"
          relevance="2.0" popularity="2" peerGroupID="urn:jxta:jxta-NetGroup" keyword="deporte"
          isLocal="false" contentName="Los deportes de raqueta" contentID="urn:jxta:uuid-
          59616261646162614E504720503250338FF7AA97421C4A0BB399A8CB98B8043007"/>
        <CodatConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-15 13:50:56.515"
          relevance="1.0" popularity="10" peerGroupID="urn:jxta:jxta-NetGroup" keyword="deporte"
          isLocal="false" contentName="Historia de la ACB" contentID="urn:jxta:uuid-
          59616261646162614E504720503250331E6AEC19C540463F87E5D9DCE743D0D807"/>
      </Keyword>
    </PeerGroup>
  - <PeerGroup peerGroupName="Asociacion cicloturistas" id="urn:jxta:uuid-
    41736F6369614369AF6EA06369636C6F59616261646162614E5047205032503302">
    - <Keyword value="rutas">
        <CodatConfidence peerGroupName="Asociacion cicloturistas" timestamp="2012-01-19
          13:07:34.301" relevance="1.61875" popularity="2" peerGroupID="urn:jxta:uuid-
          41736F6369614369AF6EA06369636C6F59616261646162614E5047205032503302"
          keyword="rutas" isLocal="false" contentName="Camino de Santiago" contentID="urn:jxta:uuid-
          41736F6369614369AF6EA06369636C6FDAC66D5400CE48BCA3A48DAEAC9D07438007"/>
      </Keyword>
    - <Keyword value="conciertos">
        <CodatConfidence peerGroupName="Asociacion cicloturistas" timestamp="2012-01-19
          13:09:26.376" relevance="1.674859375" popularity="1" peerGroupID="urn:jxta:uuid-
          41736F6369614369AF6EA06369636C6F59616261646162614E5047205032503302"
          keyword="conciertos" isLocal="false" contentName="Gira RCHP 2012" contentID="urn:jxta:uuid
          -436C75622066416EB320A465205243482FA4A3E0CA8242B8937EB49FE9D26B758007"/>
      </Keyword>
    </PeerGroup>
</ContentConfidenceTable>
```


Figura 28: Fichero XML de una Tabla de Confianza en Contenido

- **Tabla de confianza en *peers* dependiente de grupo.**

Al igual que en el caso anterior, se recurrirá al punto de vista del sistema de archivos en el que, como recordaremos, estas tablas estarán almacenadas en un fichero *XML*. Este fichero contendrá un elemento (o etiqueta) único de nivel superior que se denominará con el nombre `<PeerConfidenceTable>`. Esta etiqueta podrá contener uno o varios elementos de tipo `<PeerGroup>` uno por cada grupo al que pertenezca el *peer*. Cada uno de estos elementos contendrá uno o más etiquetas de tipo `<Keyword>` y estos a su vez uno o más elementos de tipo `<PeerConfidence>`. Se puede ver esta estructura con el ejemplo mostrado en la figura 29.

```
<?xml version="1.0" encoding="UTF-8"?>
- <PeerConfidenceTable>
  - <PeerGroup peerGroupName="NetPeerGroup" id="urn:jxta:jxta-NetGroup">
    - <Keyword value="moda">
        <PeerConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-15 12:30:56.515"
          sumContentConf="15.36" popularity="55" peerName="Bot2" peerID="urn:jxta:uuid-
          59616261646162614E50472050325033633 7F6DAAE0F483AB8C1FA5CDE3B3F9003"
          peerGroupID="urn:jxta:jxta-NetGroup" numContentConf="10" keyword="moda"
          confidenceValue="2.01"/>
        <PeerConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-15 13:40:56.515"
          sumContentConf="1.0" popularity="42" peerName="Bot3" peerID="urn:jxta:uuid-
          59616261646162614E504720503250333FED9CFC9D534214B064CCF55596F55903"
          peerGroupID="urn:jxta:jxta-NetGroup" numContentConf="1" keyword="moda"
          confidenceValue="3.99"/>
      </Keyword>
    - <Keyword value="java">
        <PeerConfidence peerGroupName="NetPeerGroup" timestamp="2011-04-15 13:40:56.515"
          sumContentConf="2.63" popularity="24" peerName="ww" peerID="urn:jxta:uuid-
          59616261646162614E50472050325033970D9F305B2C491BBCC1AE3322D0C80103"
          peerGroupID="urn:jxta:jxta-NetGroup" numContentConf="1" keyword="java"
          confidenceValue="1.58"/>
      </Keyword>
    </PeerGroup>
  - <PeerGroup peerGroupName="Club fans de RCHP" id="urn:jxta:uuid-
    436C75622066416EB320A4652052434859616261646162614E5047205032503302">
    - <Keyword value="conciertos">
        <PeerConfidence peerGroupName="Club fans de RCHP" timestamp="2012-01-19 13:00:54.556"
          sumContentConf="1.5" popularity="1" peerName="ee" peerID="urn:jxta:uuid-
          59616261646162614E5047205032503314371760637240119C18B7CD63D5C72503"
          peerGroupID="urn:jxta:uuid-
          436C75622066416EB320A4652052434859616261646162614E5047205032503302"
          numContentConf="1" keyword="conciertos" confidenceValue="1.25"/>
        <PeerConfidence peerGroupName="Club fans de RCHP" timestamp="2012-01-19 13:09:26.362"
          sumContentConf="0.9225" popularity="1" peerName="qq" peerID="urn:jxta:uuid-
          59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203"
          peerGroupID="urn:jxta:uuid-
          436C75622066416EB320A4652052434859616261646162614E5047205032503302"
          numContentConf="1" keyword="conciertos" confidenceValue="0.9612499999999999"/>
      </Keyword>
    </PeerGroup>
</PeerConfidenceTable>
```

Figura 29: Fichero XML de una Tabla de Confianza en Peer

- **Tabla de confianza en *peers* independiente de grupo.**

Desde el punto de vista de almacenamiento, al igual que ocurría con las tablas anteriores, la de confianza en *peer* independiente de grupo será un nuevo fichero *XML*. Este fichero tendrá un único elemento de nivel superior con la etiqueta `<PeerConfidenceTableGI>` que contendrá uno o más elementos de tipo `<Keyword>`. Cada uno de ellos, a su vez, contendrá uno o más elementos de tipo `<PeerConfidence>`. Un ejemplo de este fichero se puede ver en la figura 30.

```
<?xml version="1.0" encoding="UTF-8"?>
- <PeerConfidenceTableGI>
  - <Keyword value="rutas">
      <PeerConfidence timestamp="2012-01-19 13:07:34.301" sumContentConf="3.11875" popularity="2"
        peerName="qq" peerID="urn:jxta:uuid-
        59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203"
        numContentConf="2" keyword="rutas" confidenceValue="1.4046875"/>
    </Keyword>
  - <Keyword value="conciertos">
      <PeerConfidence timestamp="2012-01-19 13:08:43.79" sumContentConf="2.9" popularity="1"
        peerName="ee" peerID="urn:jxta:uuid-
        59616261646162614E5047205032503314371760637240119C18B7CD63D5C72503"
        numContentConf="1" keyword="conciertos" confidenceValue="1.95"/>
    </Keyword>
  - <Keyword value="deporte">
      <PeerConfidence timestamp="2010-04-15 13:30:56.515" sumContentConf="8.0" popularity="44"
        peerName="Bot2" peerID="urn:jxta:uuid-
        59616261646162614E50472050325033633 7F6DAAE0F483AB8C1FA5CDE3B3F9003"
        numContentConf="4" keyword="deporte" confidenceValue="3.54"/>
      <PeerConfidence timestamp="2011-04-02 13:30:56.515" sumContentConf="15.36" popularity="22"
        peerName="Bot3" peerID="urn:jxta:uuid-
        59616261646162614E504720503250333FED9CFC9D534214B064CCF55596F55903"
        numContentConf="10" keyword="deporte" confidenceValue="1.66"/>
      <PeerConfidence timestamp="2012-01-25 19:21:34.734" sumContentConf="1.8699999999999999"
        popularity="1" peerName="qq" peerID="urn:jxta:uuid-
        59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203"
        numContentConf="1" keyword="deporte" confidenceValue="1.435"/>
    </Keyword>
</PeerConfidenceTableGI>
```

Figura 30: Fichero XML de una Tabla de Confianza en Peer indep. de Grupo

- **Tabla de riesgo**

El fichero *XML* correspondiente a esta tabla también es bastante sencillo. Se compone de un elemento único de nivel superior etiquetado como `<RiskTable>` que contiene uno o más elementos de tipo `<Risk>`. Estos elementos contendrán los atributos necesarios para definir completamente un valor de riesgo, a saber, el identificador y el nombre del *peer* al que hace referencia, y los valores de accesibilidad, integridad y rendimiento del mismo. Se puede ver en la figura 31.

```
<?xml version="1.0" encoding="UTF-8"?>
- <RiskTable>
    <Risk performance="4.0" peerName="ee" peerID="urn:jxta:uuid-
      59616261646162614E5047205032503314371760637240119C18B7CD63D5C72503"
      integrity="2.0" accesibility="11.0"/>
    <Risk performance="3.9375" peerName="ww" peerID="urn:jxta:uuid-
      59616261646162614E50472050325033970D9F305B2C491BBCC1AE3322D0C80103"
      integrity="2.0" accesibility="6.0"/>
    <Risk performance="1.6" peerName="Bot4" peerID="urn:jxta:uuid-
      59616261646162614E504720503250334 6C2B17250A04857BD81D38E3CA1E05903"
      integrity="3.2" accesibility="1.1"/>
    <Risk performance="3.9" peerName="Bot3" peerID="urn:jxta:uuid-
      59616261646162614E504720503250333FED9CFC9D534214B064CCF55596F55903" integrity="3.9"
      accesibility="3.6"/>
    <Risk performance="2.2" peerName="Bot1" peerID="urn:jxta:uuid-
      1FE591D313C3415598E5B97608096B0A6D84479024A241C5A9BE9AA34B17E74F03"
      integrity="1.2" accesibility="1.2"/>
</RiskTable>
```

Figura 31: Fichero XML de una Tabla de Riesgo

3.3.3. Anuncio de peer

Tal y como se ha explicado previamente, este sistema de gestión de la reputación aprovechará las posibilidades ofrecidas por la plataforma *JXTA*, concretamente la publicación de anuncios para cada uno de los *peers*. Estos anuncios servirán para que el usuario se dé a conocer dentro de la red, ofreciendo sus contenidos locales.

El anuncio básico de un *peer* es una estructura de datos *XML* que contiene toda la información necesaria para ponerse en contacto con el mismo, desde su identificador único hasta la dirección a través de la cual podrá recibir peticiones. Este anuncio es generado automáticamente por el gestor de red, descrito en la clase `Peer`, y tiene un aspecto similar a este:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE jxta:PA (View Source for full doctype...)>
- <jxta:PA xml:space="default" xmlns:jxta="http://jxta.org">
    <PID>urn:jxta:uuid-
      59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203</PID>
    <GID>urn:jxta:jxta-NetGroup</GID>
    <Name>qq</Name>
    <Desc />
  - <Svc>
      <MCID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE0000000805</MCID>
    - <Parm>
      - <jxta:RA xml:space="preserve" xmlns:jxta="http://jxta.org">
          <DstPID>urn:jxta:uuid-
            59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203</DstPID>
        - <Dst>
          - <jxta:APA xmlns:jxta="http://jxta.org" xml:space="preserve">
              <EA>tcp://192.168.1.10:9704</EA>
              <EA>tcp://192.168.50.1:9704</EA>
              <EA>relay://uuid-
                59616261646162614E504720503250337BE50CD9D50F4D4299CF9C9816C00FC203</EA>
            </jxta:APA>
          </Dst>
        </jxta:RA>
      </Parm>
    </Svc>
</jxta:PA>
```

Figura 32: Anuncio de Peer original

Este anuncio, desde el punto de vista de un sistema de gestión de la reputación, no tiene mucha utilidad si no es estrictamente para facilitar la conectividad del mismo con otros usuarios, y por lo tanto no debería mencionarse como estructura fundamental del sistema. Desde este punto de vista se estaría desaprovechando el potencial de un elemento que llegará a todos los *peers* de la red.

Por este motivo se ha decidido utilizar esta estructura para difundir un índice de los contenidos almacenados en las tablas locales de confianza de cada *peer*. Para ello se debe modificar el anuncio original, añadiéndole dicha información.

De esta manera se aprovechará el elemento `<Desc>`, presente en el anuncio como un elemento vacío, y que no se utilizará para otros fines. En este preciso elemento se insertará toda la información de los contenidos, tal y como se puede ver en la figura 33.

```
- <Desc>
  - <ContentConfidenceTable>
    - <PeerGroup peerGroupName="NetPeerGroup" id="urn:jxta:jxta-NetGroup">
        <Keyword value="hogar" />
        <Keyword value="moda" />
        <Keyword value="deporte" />
      </PeerGroup>
    - <PeerGroup peerGroupName="Asociacion cicloturistas" id="urn:jxta:uuid-
      41736F6369614369AF6EA06369636C6F59616261646162614E5047205032503302">
        <Keyword value="rutas" />
        <Keyword value="conciertos" />
      </PeerGroup>
    - <PeerGroup peerGroupName="Grupo Prueba 1" id="urn:jxta:uuid-
      99E74F27B58F4A19B044EA2D7BC548EB02">
        <Keyword value="motor" />
        <Keyword value="fauna" />
      </PeerGroup>
    </ContentConfidenceTable>
  </Desc>
```

Figura 33: Detalle elemento <Desc> del anuncio de Peer

Como se puede comprobar, en la descripción del anuncio se incluirá información referente a todos los grupos a los que pertenece el *peer*, así como todas las palabras clave para las que tiene contenidos en sus tablas. Esto se hace insertando un elemento de tipo `<ContentConfidenceTable>` que podrá tener uno o varios elementos hijos de tipo `<PeerGroup>`, uno por cada grupo al que pertenezca el *peer*. Cada elemento `<PeerGroup>` tendrá a su vez uno o varios elementos de tipo `<Keyword>` correspondientes a las diferentes palabras clave para cada grupo.

Cada elemento de tipo `<PeerGroup>` tendrá a su vez dos atributos que indicarán el nombre del grupo y su identificador *JXTA* único. Por otro lado cada elemento `<Keyword>` tendrá un atributo que representará el valor de la palabra clave.

Con esta información, cualquier nodo que reciba este anuncio podrá conocer de primera mano la información por la que puede preguntar a un *peer* determinado, así como la dirección a la que debe dirigir dichas peticiones.

3.4. Desarrollos algorítmicos no triviales

Para completar la descripción del sistema y poder entender su funcionamiento por completo necesitamos abordar la actualización de los valores de reputación, que nos permitirán tener actualizadas las tablas descritas en la sección anterior, así como aquellos necesarios para el cálculo del riesgo para interactuar con un *peer* determinado.

Antes de comenzar con su descripción se debe mencionar que, mientras los cálculos para la actualización de los valores de reputación y los correspondientes a la obtención de los umbrales de cooperación son los originales de *Poblano* [26] o presentan muy pocos cambios respecto a estos, la elección del *peer* al que se realizará la petición es un algoritmo completamente nuevo.

3.4.1. Actualización de los valores de reputación

La actualización de los valores de reputación es, hasta cierto punto, un proceso complejo ya que cada *peer* necesita actualizar tres tipos de tabla de confianza. Del mismo modo, las actualizaciones se basan tanto en las calificaciones de los *peers* como en los valores de reputación recibidos a través de *feedbacks*. En este apartado se describirán cada uno de estos modelos de actualización, comenzando con el más simple de todos:

- **Caso 1:**

Un *peer* necesita actualizar un valor antiguo de reputación, denotado como `ContentConf`, utilizando su propia calificación de un contenido así como los valores `ContentConf` propagados por otros *peers* remotos. La popularidad de estos `ContentConf` propagados será la media aritmética de todos ellos. La función de actualización es:

$$ContentConf_{new} = F(ContentConf_{old}, ContentConf_{propagated}, userRating)$$
$$= (a \times ContentConf_{old}) + (b \times ContentConf_{propagated})$$
$$+ (c \times userRating)$$

Las variables *a*, *b*, y *c* deben ser números reales no negativos. También se debe cumplir que la suma de las tres debe ser igual a uno. Por último el valor de *c* debe ser siempre 0.70 ya que el criterio del usuario debe ser el que prevalezca sobre los demás parámetros.

En cuanto a los valores que pueden tomar las variables *a* y *b*, estos dependerán de la popularidad de los contenidos correspondientes. Si la popularidad de los contenidos propagados es superior a la del antiguo valor de reputación se deberá utilizar *a*=0.10 y *b*=0.20. En caso contrario se utilizará *a*=0.20 y *b*=0.10.

En este caso, el parámetro `userRating` será una entrada proporcionada por el usuario. Puede ocurrir en algún caso que no esté disponible ni $\texttt{ContentConf}_{\texttt{old}}$ ni $\texttt{ContentConf}_{\texttt{propagated}}$ por lo que deberán ser prefijados con valor 1. Esta misma regla puede ser aplicada a las funciones descritas a continuación.

- **Caso 2:**

Un *peer* desea actualizar un valor de reputación antiguo utilizando el *feedback* recibido de otro usuario remoto. Aunque no siempre ocurre, el usuario puede tener almacenado en sus tablas de confianza en *peer* una reputación de *peer*, o `PeerConf` correspondiente al *peer* que envió el *feedback*, pudiendo utilizarlo en la siguiente función de actualización:

$$ContentConf_{new} = F\left(ContentConf_{old}, feedback, PeerConf_{feedbacker}\right) = \frac{\left(ContentConf_{old} + \left(feedback \times \frac{PeerConf_{feedbacker}}{4}\right)\right)}{2}$$

En la mayoría de los casos, el usuario no tiene un valor de `PeerConfidence` para el *feedbacker*, por lo que este parámetro puede ser prefijado a 1.

- **Caso 3:**

Un *peer* desea actualizar un valor de reputación de *peer*, o `PeerConf`, correspondiente a un nodo proveedor de información dentro de un grupo determinado. Aquí, nadie le dice al *peer* local cómo es el rendimiento del usuario remoto, por lo que este debe generar su propia opinión sobre el proveedor, asociada a una o varias palabras clave. Lo que el *peer* sí conoce son todos los `ContentConf` generados a partir de contenidos que el peer remoto ha enviado. Por lo tanto, la función de actualización será:

$$PeerConf_{new} = F\left(PeerConf_{old}, ContentConf_{relacionados\ con\ el\ proveedor}\right) = \frac{\left(PeerConf_{old} + \frac{1}{|K|}\sum_{a \in K} ContentConf_{proveedor}\right)}{2}$$

Donde |K| es el número de palabras clave relacionadas con el proveedor.

3.4.2. Cálculo del riesgo

El riesgo, como se vio en el apartado correspondiente a *Poblano* dentro del capítulo del Estado del Arte, trata de establecer una medida objetiva de la calidad de un *peer*, atendiendo a criterios de accesibilidad, rendimiento e integridad de los datos compartidos.

Cada vez que se realice un intercambio de información con un *peer*, el valor de todos estos factores será actualizado de acuerdo con el resultado de dicha interacción, y el resultado será un valor dentro del rango [0,4].

El método de cálculo del factor de riesgo en base a los tres componentes que lo definen se puede ver en la siguiente fórmula:

$$Risk = 4 - \frac{accesibility + performance + integrity}{3}$$

Para completar la definición de esta actualización de los valores de riesgo, se debe mencionar cómo irán evolucionando cada uno de los componentes por separado:

- Accesibilidad (*accesibility*): Cada vez que se intente contactar con un *peer* y este envíe una respuesta, se considerará que la accesibilidad del *peer* es buena, y por lo tanto se sumará uno al valor que tuviese antes, hasta un

máximo de 4. En caso contrario se restará una unidad a este valor, hasta un valor mínimo de -1.

- Rendimiento (*performance*): Es una medida del tiempo de respuesta del *peer* cada vez que se contacta con él, que se traducirá, al igual que la accesibilidad, en un valor comprendido en el rago [-1, 4].
- Integridad (*integrity*): Esta es una medida que evalúa la calidad de los contenidos compartidos por dicho *peer*, por lo que será el valor de reputación de dichos contenidos en el sistema local e irá evolucionando de la misma forma.

3.4.3. Cálculo de umbrales de cooperación

Para hacer todos estos valores de reputación útiles para los usuarios se introduce el concepto de umbral de cooperación. Si un valor de reputación en *peer* es mayor que dicho umbral podemos considerar a dicho *peer* como dispuesto a cooperar. En otro caso, la interacción con dicho *peer* resultaría demasiado arriesgada para el usuario.

El cálculo del umbral de cooperación está basado en el factor de riesgo, los valores `ContentConfidence` y un valor de importancia. Este concepto de importancia es nuevo y es una medida de lo importante que es dicha cooperación para el usuario. Puede tener un valor de (-1, 0, 1, 2, 3, 4), siendo su valor por defecto 2.

El valor del factor de riesgo está comprendido entre 0 y 4 y es el resultado de una simple operación sobre los valores de integridad, rendimiento y accesibilidad de dicho *peer*. Si este valor no se conoce se tomará como valor por defecto el 2. Así, la cooperación será posible si:

$$PeerConf_{Keyword} \times Importance > \frac{Risk_{Peer}}{\frac{1}{|K|}\sum_{a \in K} ContentConf_{Peer}}$$

A efectos prácticos en el desarrollo de la aplicación se ha previsto realizar un cambio en la desigualdad anterior para obtener un único valor numérico. Este cambio implica la resta del segundo término al primero, lo que nos dará un valor del grado de posible cooperación del *peer* remoto. De esta forma, si para un nodo obtenemos un valor positivo, ya podremos suponer que estará dispuesto a colaborar, en caso contrario la colaboración resultaría demasiado arriesgada.

Cuando se realiza el cálculo de un umbral de cooperación no se efectúa para un único *peer*, sino que el cálculo se hace para un conjunto de ellos con el objetivo de elegir al mejor (con una determinada probabilidad). El hecho de tener este único valor numérico permite realizar una ordenación de los mismos de mayor a menor predisposición a la cooperación.

3.4.4. Elección de un peer

Se ha visto que el método principal para determinar si un *peer* estará dispuesto a colaborar con el sistema local y enviar contenidos de buena calidad es determinar su umbral de cooperación. Este método resulta correcto pero tiene un inconveniente: Con total seguridad todas las peticiones de contenidos se enviarán a un conjunto reducido de los *peers* que tengan mejores valores de reputación dentro del sistema, es decir, los que más confianza le inspiren a otros nodos, lo que podría desembocar en una saturación de los mismos y hacer al sistema vulnerable a ataques de tipo denegación de servicio.

Para evitar este problema, en nuestro sistema se ha aplicado una función probabilística, configurable por el usuario, que permitirá determinar el *peer* que se elegirá finalmente para interactuar con él. Esto se realizará gracias a un parámetro denominado *pBest*.

Una vez se ha obtenido el conjunto de los *peers* candidatos para proporcionar una determinada información se realizará una ordenación en base a su umbral de cooperación, tal y como se ha visto en el apartado anterior.

Tras ser ordenados, *pBest* indicará la probabilidad de que el *peer* con mejor grado de cooperación sea elegido finalmente para realizar la interacción. El sistema generará un número aleatorio, que denotaremos como *p*, que permitirá seleccionar a uno de los *peers* de la tabla anterior basándose en su posición dentro de ella.

Si $0 < p < pBest$ se seleccionará directamente el primer candidato de la lista ordenada, es decir, el *peer* en el que se tiene más reputación. Por otro lado si $pBest < p < 1$ deberemos seleccionar con igual probabilidad a uno de los *peers* restantes, por lo que deberemos aplicar un algoritmo que nos devuelva directamente el índice del *peer* seleccionado. Así, el índice de la tabla elegido será:

$$\left[\frac{(p - pBest) \times (K - 1)}{(1 - pBest)} + 2\right]$$

En esta expresión los corchetes indican la función parte entera del número resultante de la operación interna, y *K* el número de *peers* presentes en la lista.

CAPÍTULO 4: IMPLEMENTACIÓN DEL SISTEMA

En este capítulo se irá un paso más allá, intentando entender mejor el funcionamiento del sistema mediante la explicación de cada uno de los módulos que lo componen. Para ello, en primer lugar se dará una visión global de todos los módulos como integrantes de un conjunto, y después se dará información detallada de la estructura y utilidad de cada una de las clases que participan en el sistema de forma independiente.

En los siguientes apartados se tratarán las clases que forman el sistema independientemente de los interfaces que implementa cada una de ellas. De esta forma tendremos el siguiente listado de clases que se irán desarrollando a continuación:

- `ContentConfidence`
- `ContentConfidenceTable`
- `PeerConfidence`
- `PeerConfidenceTable`
- `PeerConfidenceTableGI`
- `Risk`
- `RiskTable`
- `TableManager`
- `PLogger`
- `PMessage`
- `Peer`
- `Autosave`

El orden en el que se abordarán estos apartados se ha dispuesto de forma que sea más fácil comprender la relación entre las clases y los módulos, y que se vea claramente cómo interactúan. Del mismo modo, se ha hecho una clasificación que permitirá identificar fácilmente las clases que forman parte de cada uno de los módulos.

Para más información referente a métodos y atributos de estas clases se puede consultar el anexo correspondiente al manual del desarrollador, en el que se hará una presentación detallada de cada uno de ellos.

4.1. Sistema completo

A lo largo de las secciones siguientes se explicarán todas y cada una de las clases que conforman el sistema de forma independiente, lo que permitirá conocer la estructura de todas ellas, dejando entrever cómo se relacionan.

Antes de hacer este análisis minucioso, llega el momento de presentar una visión general de la estructura del sistema de gestión de reputación, a través del diagrama de clases recogido en la figura 34.

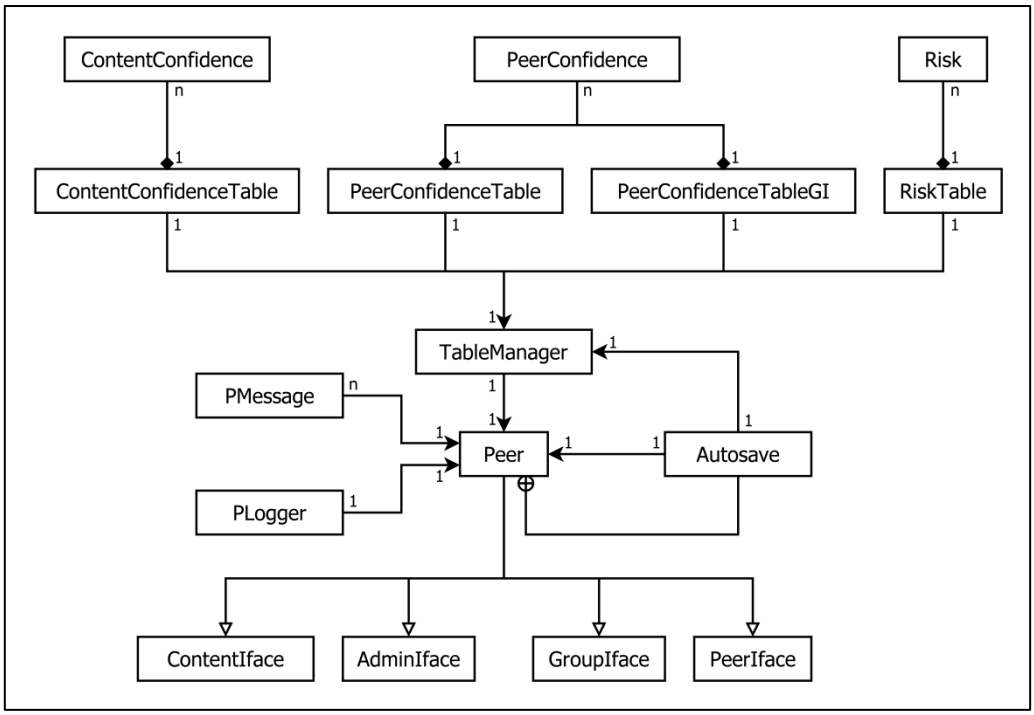


Figura 34: Diagrama UML del sistema completo

A través del diagrama de clases se puede observar qué clases dependen de otras y en qué medida. Así, la clase `Peer` estará en el centro del sistema, representando en nexo de unión de todas ellas. Esta clase `Peer` estará formada principalmente por un elemento de la clases `TableManager`, que le dará acceso al módulo de gestión de tablas. Del mismo modo esta clase implementará los interfaces `ContentIface`, `AdminIface`, `GroupIface` y `PeerIface` que se verán en la siguiente sección.

Ya que se va a realizar un repaso por todas y cada una de las clases que componen el sistema, se debe volver la vista atrás, y recordar que el diseño del mismo se basa en la existencia de cuatro módulos fundamentales, el de gestión de las tablas, el gestor de *logs*, el gestor de la comunicación y el de gestión de la reputación propiamente dicho.

4.2. Descripción de las interfaces

Las interfaces muestran cómo el usuario, o cualquier aplicación externa, será capaz de interaccionar con el sistema, definiendo un conjunto limitado de métodos que él mismo podrá invocar directamente y que conformarán el núcleo de funciones para la gestión de los valores de reputación en la red.

Para dar una visión más clara de estas funciones, se ha hecho una división en cuatro interfaces, que contendrán métodos agrupados en función de qué componente del sistema se podrá modificar con cada uno de ellos. Atendiendo a este criterio los interfaces resultantes son los siguientes:

- Interfaz de administrador (`AdminIface`)
- Interfaz de contenido (`ContentIface`)
- Interfaz de *peer* (`PeerIface`)

- Interfaz de grupo (`GroupIface`)

En la figura 35 se puede ver la estructura del sistema en capas, posicionando a los interfaces como un estrato intermedio que permitirá a las aplicaciones acceder a toda la funcionalidad del sistema de gestión de la reputación.

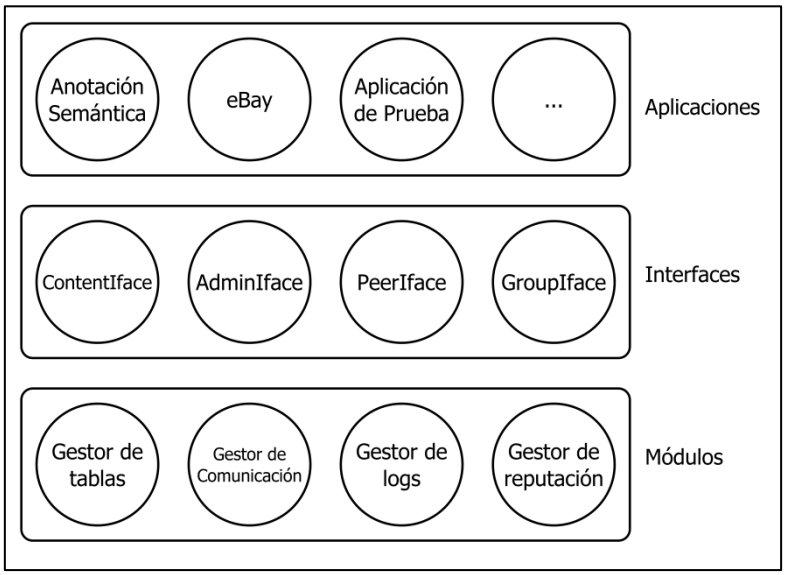


Figura 35: Estructura en capas del sistema

4.2.1. *Interfaz de administrador*

Empezaremos con el interfaz que menos tiene que ver con el funcionamiento de los mecanismos de gestión de la reputación, pero que también tiene su importancia dentro del sistema. Este contendrá todos los métodos necesarios para la configuración general del sistema, centralizando de este modo toda la gestión de las propiedades que, aún sin estar ligados directamente con la gestión de la reputación, dan un soporte complementario que hace que todo funcione correctamente.

Podemos decir que este interfaz está pensado para un usuario avanzado del sistema, por ejemplo un administrador, que no se ciña a la búsqueda y evaluación de contenidos, sino que se encargue de labores de supervisión y configuración, verificando que todo funcione correctamente.

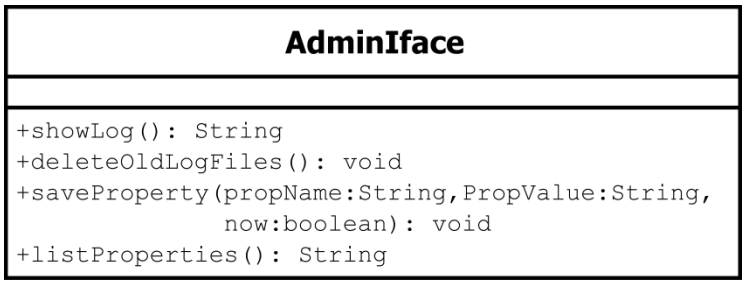


Figura 36: Diagrama UML del interfaz AdminIface

Este interfaz proporcionará los mecanismos necesarios para la gestión de los registros de *log* que se generarán a lo largo del ciclo de vida del sistema. En concreto, se permitirá al usuario mostrar las entradas más recientes de dichos registros así como eliminar los ficheros correspondientes a las sesiones más antiguas.

Por otro lado, y siguiendo con las funciones de administrador, este interfaz permitirá consultar y modificar los valores almacenados en el fichero de configuración, que regirán el comportamiento del sistema, y que darán la posibilidad a un usuario avanzado de estudiar cómo cambios en dichas variables afectan al sistema.

4.2.2. Interfaz de contenido

El siguiente de los interfaces va a permitir realizar toda la gestión de los contenidos y sus valores de reputación, desde su búsqueda hasta su eliminación. Se podría decir que este es el interfaz que recoge la funcionalidad básica sin la cual el sistema no tendría sentido (Figura 37).

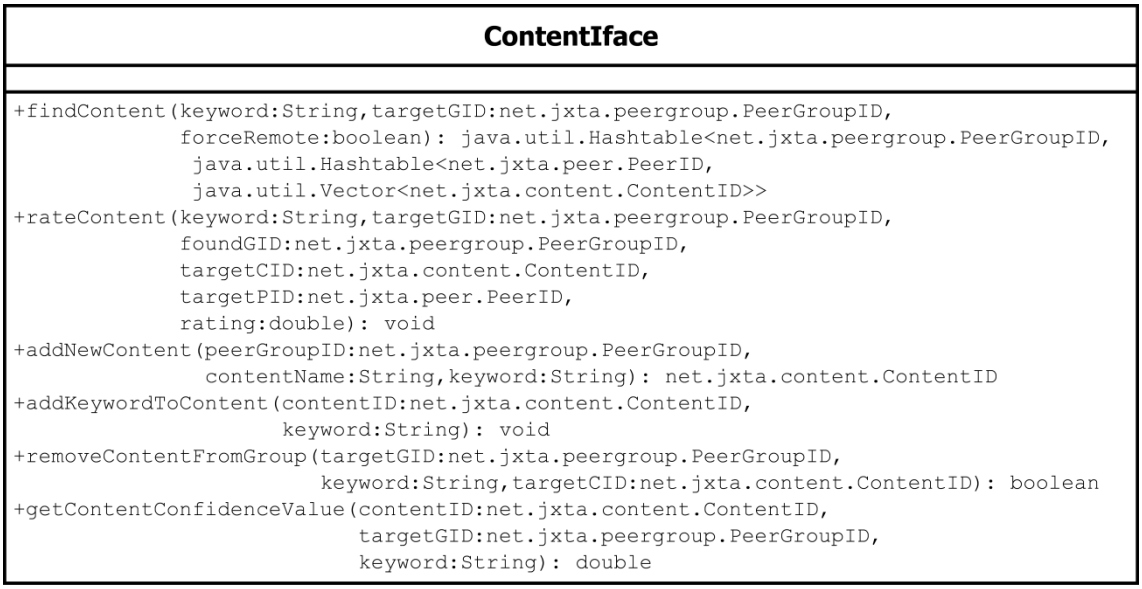


Figura 37: Diagrama UML del interfaz ContentIface

Mediante la implementación de este interfaz el usuario será capaz de gestionar las tablas locales de confianza en contenidos de forma directa, así como realizar las operaciones de evaluación correspondientes tras cada consulta.

Cada uno de los métodos se encarga de realizar una operación básica con un contenido, de forma que se ofrecerá al usuario la posibilidad de:

- Buscar un contenido local o en la red.
- Evaluar un contenido consultado previamente.
- Añadir un nuevo contenido.
- Etiquetar un contenido con una nueva palabra clave.
- Eliminar un contenido.
- Consultar el valor de reputación de un contenido.

4.2.3. *Interfaz de peer*

El núcleo fuerte de las operaciones que permite el sistema, como hemos visto en el interfaz anterior, van dirigidas a los contenidos. A pesar de esto, para completar la funcionalidad del sistema deberá estar dotado de los mecanismos necesarios para la gestión de los *peers* con los que tendrán que establecerse las comunicaciones.

Por ello se hace necesaria la existencia de un interfaz nuevo que recoja estos métodos tan necesarios para el correcto funcionamiento del sistema. En la figura 38 se puede ver que agrupa tanto métodos de actualización de información, que se encargan de modificar valores en las tablas de confianza, como otros que se limitan a la consulta de dichos valores.

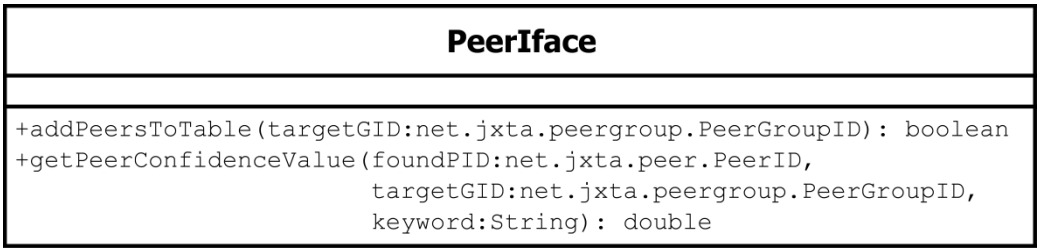


Figura 38: Diagrama UML del interfaz PeerIface

A pesar de la importancia de este interfaz de cara a la gestión de *peers* dentro de las tablas de confianza, se podrá comprobar que el número de métodos correspondientes es suficientemente pequeño para evitar la complicación innecesaria del mismo.

Así, este interfaz contará con los mecanismos necesarios para obtener el valor de reputación asignado a un peer determinado y para agregar nuevos nodos, que hasta el momento eran desconocidos, a las tablas locales del sistema.

4.2.4. *Interfaz de grupo*

Por último solo queda hacer referencia a un interfaz que, por encima de la gestión de contenidos y *peers*, se encarga del mantenimiento de los grupos de intereses creados en la red. Como se ha visto hasta el momento, los grupos son la base del intercambio y búsqueda de información en la red, permitiendo realizar consultas más precisas y obtener resultados más ajustados.

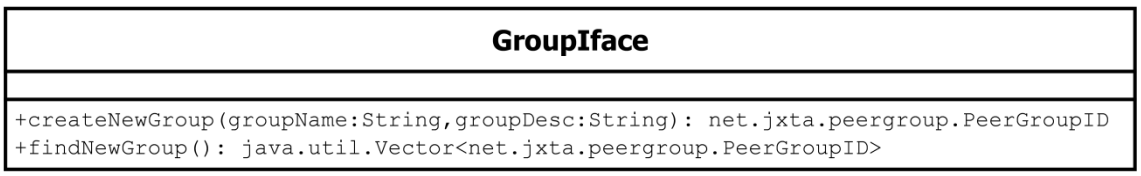


Figura 39: Diagrama UML del interfaz GroupIface

Este interfaz contiene dos funcionalidades básicas en lo que a gestión de grupos se refiere, y que abordan todas las operaciones que el sistema puede querer efectuar con los mismos.

Por un lado, se permitirá la búsqueda de nuevos grupos de intereses que amplíen las posibilidades de éxito en cada búsqueda de contenidos que se realice. Por otro lado, si no se encuentra ningún grupo adecuado, se ofrecerá la posibilidad de crear un nuevo grupo y publicarlo en la red para que pueda ser descubierto por el resto de usuarios de la red.

4.3. Módulo de gestión de tablas

Como se puede ver en la figura 40, este módulo cuenta con más clases constituyentes, y es uno de los más importantes dentro del sistema de gestión de la reputación.

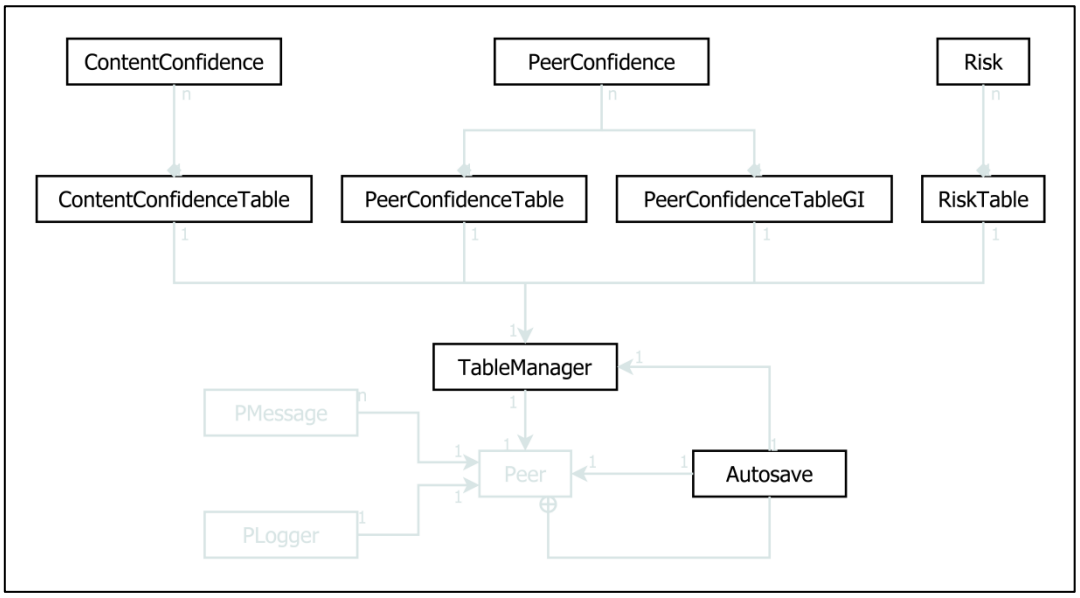


Figura 40: Módulo de gestión de tablas

4.3.1. ContentConfidence

Esta clase modela la unidad básica de medida para la reputación de contenidos que tiene el usuario, siendo el mecanismo que utiliza el sistema para que el usuario pueda determinar la calidad de un contenido dado.

El atributo fundamental de esta clase es el valor de reputación propiamente dicho, valor numérico que se va actualizando a lo largo del tiempo mediante puntuaciones subjetivas de los usuarios. Adicionalmente, y para que esta clase tenga sentido, deben añadirse atributos tales como un identificador del contenido al que hace referencia y otros que nos ayudarán a tratar con los valores de reputación. El diagrama *UML* mostrado en la figura 41 detalla al completo la estructura de la clase.

Esta estructura es fundamental para el correcto funcionamiento del sistema, ya que constituye el método de representación de la reputación en contenidos, ofreciendo tanto los atributos necesarios para su almacenamiento, como los métodos que permitirán acceder y modificar los mismos.

Entre todos estos métodos, caben destacar los correspondiente a la actualización de los valores de reputación del contenido: `updateConfValueByRate` y `updateConfValueByFeedback`.

El primero de ellos se encargará de realizar la actualización siempre que un *peer* haya accedido a un contenido y desee evaluarlo, utilizando como información adicional los valores de reputación propagados por otros nodos de la red.

El segundo de estos métodos actualiza el valor de reputación en base al *feedback* recibido de un *peer* remoto de la red que previamente habrá evaluado el contenido.

ContentConfidence

```
+keyword: String
+contentID: net.jxta.content.ContentID
+contentName: String
+peerGroupID: net.jxta.peerGroup.PeerGroupID
+peerGroupName: String
+isLocal: boolean
+popularity: int
+relevance: double
+timestamp: java.sql.Timestamp
```

```
+ContentConfidence()
+ContentConfidence(keyword:String,contentID:net.jxta.content.ContentID,
                   contentName:String,sourcePeerID:net.jxta.peer.peerID,
                   peerGroupID:net.jxta.peerGroup.PeerGroupID,
                   peerGroupName:String,isLocal:boolean,
                   popularity:int,relevance:double,
                   timestamp:java.sql.Timestamp)
+getKeyword(): String
+getContentID(): net.jxta.content.ContentID
+getContentName(): String
+getPeerGroupID(): net.jxta.peerGroup.PeerGroupID
+getPeerGroupName(): String
+isLocal(): boolean
+getPopularity(): int
+getRelevance(): double
+getTimestamp(): java.sql.Timestamp
+setKeyword(keyword:String): void
+setContentID(contentID:net.jxta.content.ContentID): void
+setContentName(contentName:String): void
+setPeerGroupID(peerGroupID:net.jxta.peerGroup.PeerGroupID): void
+setPeerGroupName(peerGroupName:String): void
+setLocal(isLocal:boolean): void
+setPopularity(popularity:int): void
+setRelevance(relevance:double): void
+setTimestamp(timestamp:java.sql.Timestamp): void
+increasePopularity(): void
+updateConfValueByRate(propagated:Vector<ContentConfidence>,
                       rating:double): void
+updateConfValueByFeedback(feedback:ContentConfidence,
                           feedbacker:PeerConfidence): void
```

Figura 41: Diagrama UML de la clase ContentConfidence

4.3.2. ContentConfidenceTable

Esta clase se encarga de estructurar todos los valores de reputación de las anotaciones que el sistema tiene almacenadas localmente. Dicha estructuración se consigue mediante una tabla en la que se ordenan todos los contenidos por grupos y por palabras clave, tal y como se presentó en la sección 3.2 del capítulo 3.

La motivación de esta clase es tener un módulo específico dentro del sistema que se encargue de la reputación de las anotaciones, esto incluye además de su organización, los métodos necesarios para su creación, búsqueda, mantenimiento y eliminación, tal y como muestra la figura 42. De esta forma, para cualquier desarrollador será más sencillo encontrar en la misma clase todas las operaciones que se puedan hacer con estos valores.

Para definir esta tabla de reputación de contenidos, es necesario contar con la tabla propiamente dicha, como un atributo de la clase, y la ruta al directorio en el que estará almacenada como una cadena de caracteres.

La tabla de reputación en contenidos, representada por el atributo `ContentConfTable`, es una estructura de tipo *Hashtable* que permitirá reproducir fielmente la estructura definida en la descripción del sistema, realizando la agrupación por palabras clave y por grupos de intereses.

Como elementos adicionales para completar la funcionalidad de esta clase, se cuenta con dos tablas auxiliares que permitirán hacer una traducción entre los nombres de los contenidos almacenados en la tabla y sus correspondientes identificadores *JXTA*, que permitirá optimizar el tratamiento de los datos.

Del mismo modo, y para un cometido similar, la clase contará con dos tablas que permitirán conocer el nombre y el correspondiente identificador de los grupos a los que pertenecen cada uno de los contenidos.

Estas tablas auxiliares se actualizarán cada vez que el *peer* intercambie información con algún otro nodo de la red. Esto es debido que toda la información nueva que se pueda almacenar en esta tabla, facilitará el correcto funcionamiento del sistema, y permitirá realizar búsquedas más precisas en las tablas.

Se puede observar que, a pesar de la existencia de las tablas auxiliares y los atributos referentes a la ubicación de la información en el disco, el núcleo de la clase lo forma un único objeto, `contentConfTable`, que será aquel sobre el que se realicen todas las operaciones necesarias para obtener y almacenar información del sistema, en lo que a reputación de contenidos se refiere.

Los métodos más importantes de esta clase, aparte de los correspondientes al almacenado en disco de la tabla y la correspondiente carga en memoria desde un fichero XML, serán los destinados a la consulta de datos de esta tabla. Así, el método más importante será `findContentConfidence`. Por un lado, este método se encargará de verificar si en las tablas existe una entrada que se corresponda con un contenido específico que se le pasará como parámetro, devolviendo el valor de reputación correspondiente en caso afirmativo.

Por otro lado, el método `findContentConfidence` permitirá obtener un listado con toda la información acerca de los contenidos que se ajustan a un patrón de búsqueda dado. Si se encuentran resultados, las entradas serán devueltas en forma de un listado sobre el que se podrá afinar más la búsqueda. Como se puede suponer, este método será el principal responsable de que, cuando un *peer* realiza una búsqueda de contenidos en la red, este obtenga resultados útiles.

Además de estos métodos, existen otros dos métodos fundamentales, que serán los dedicados a actualizar los valores de reputación de los contenidos de las tablas, en consonancia con los dos métodos vistos en la clase `ContentConfidence`. Por un lado, servirán para actualizar una reputación en base a una calificación local, mientras que por otro lado, actualizará dicho valor debido a la recepción de un *feedback*.

Entre los métodos auxiliares de esta clase, se pueden mencionar los correspondientes a la eliminación de contenidos de la tabla de reputación, así como aquellos destinados a realizar la traducción nombre-identificador tanto de contenidos como de grupos.

ContentConfidenceTable

```
+contentConfTable: Hashtable<net.jxta.peerGroup.PeerGroupID, Hashtable<String, Hashtable<net.jxta.content.ContentID, ContentConfidence>>>
+userPath: String
+fileName: String
+userFile: java.io.File
+contentIDTable: java.util.Hashtable<net.jxta.content.ContentID, String>
+contentNameTable: java.util.Hashtable<String, net.jxta.content.ContentID>
+groupIDTable: java.util.Hashtable<net.jaxta.peerGroup.PeerGroupID, String>
+groupNameTable: java.util.Hashtable<String, net.jxta.peerGroup.PeerGroupID>
```

```
+ContentConfidenceTable(userPath:String)
+setUserPath(newUserPath:String): void
+findContentConfidence(peerGroupID:net.jxta.peerGroup.PeerGroupID,
        contentConf:ContentConfidence): boolean
+findContentConfidence(contentConf:ContentConfidence): Vector<ContentConfidence>
+getContentConfidence(contentConf:ContentConfidence): ContentConfidence
+setContentConfidence(contentConf:ContentConfidence): boolean
+addContentConfidence(peerGroupID:net.jxta.peerGroup.PeerGroupID,
        contentConf:ContentConfidence): boolean
+updateContentConfidence(target:ContentConfidence,
        propagated:Vector<ContentConfidence>,
        rating:double): ContentConfidence
+updateContentConfidence(target:ContentConfidence,
        feedback:ContentConfidence,
        feedbackerConf:PeerConfidence): ContentConfidence
+getContentConfidenceValue(peerGroupID:net.jxta.peerGroup.PeerGroupID,
        contentID:net.jxta.content.ContentID): ContentConfidence
+addGroupTable(peerGroupID:net.jxta.peerGroup.PeerGroupID,
        pght:java.util.Hashtable<String,
        java.util.Hashtable<net.jxta.content.ContentID,
        ContentConfidence>>): boolean
+getKeywordsForAdv(): org.w3c.dom.Document
+getAvailableGroups(): java.util.Enumeration<net.jxta.peergroup.PeerGroupID
+removeContent(peerGroupID:net.jxta.peergroup.PeerGroupID,
        keyword:String,contentID:net.jxta.content.ContentID): boolean
+getContentName(contentID:net.jxta.content.ContentID): String
+getContentID(contentName:String): net.jxta.content.ContentID
+getGroupName(peerGroupID:net.jxta.peergroup.PeerGroupID): String
+getGroupID(groupName:String): net.jxta.peergroup.PeerGroupID
+addContentPair(contentID:net.jxta.content.ContentID,
        contentName:String): boolean
+addPeerGroupPair(peerGroupID:net.jxta.peergroup.PeerGroupID,
        peerGroupName:String): boolean
+toXMLFile(): void
+fromXMLFile(): void
```

Figura 42: Diagrama UML de la clase ContentConfidenceTable

4.3.3. PeerConfidence

Siguiendo con las clases fundamentales del sistema encontramos `PeerConfidence`. Esta clase es la que modela el grado de reputación que tiene el sistema en un determinado *peer* remoto, y será de mucha utilidad a la hora de elegir a quién pedir determinada información, con quién interactuar en todo momento, e incluso valorar la información recibida de otro nodo de la red.

Al igual que ocurre con la clase `ContentConfidence`, el atributo principal de esta clase será un valor numérico decimal en el rango [-1,4] que representará la medida de dicha reputación. Por otro lado, también habrá atributos, que podríamos calificar de secundarios, que nos ayudará tanto a identificar el *peer* al que hace referencia dicho valor como otros que nos ayudarán a su cálculo y actualización.

Esta estructura merece una especial atención, ya que permitirá representar la información contenida en las dos tablas que se explicarán en los apartados siguientes, por lo que tendrá una doble funcionalidad. De esta forma, en función de la tabla a la que se esté haciendo referencia, se utilizarán unos u otros atributos de la clase, lo que la convertirá en una estructura muy versátil dentro del sistema. El conjunto completo de métodos y atributos de la clase puede verse en la figura 43.

Como elementos principales, están los correspondientes a identificar unívocamente al *peer* al que hace referencia la reputación, es decir, su identificador *JXTA* único, su nombre, el grupo y la palabra clave a las que hace referencia el valor, y por supuesto, la reputación propiamente dicha.

Entre los métodos más importantes de la clase, como ocurría con `ContentConfidence`, se encuentra el destinado a la actualización de los valores de reputación de un *peer*. Este método basará sus cálculos en la reputación del último contenido recibido por el *peer* en cuestión, siendo esta la única fuente para actualizar estos valores. Esto evitará posibles manipulaciones en el mecanismo de actualización, proporcionando valores fidedignos.

Además de este método, se pueden encontrar otros métodos triviales, como por ejemplo los que permiten acceder o modificar el valor de cada uno de los atributos de la clase; o los constructores, que se encargan de inicializar todos estos atributos con una serie de valores por defecto, como es el caso del constructor básico, o con un conjunto de valores pasados como parámetro.

Figura 43: Diagrama UML de la clase PeerConfidence

4.3.4. *PeerConfidenceTable*

Los valores de reputación de los *peers* no sirven de nada sin una organización estructurada que permita recuperar información mediante búsquedas concretas. Del mismo modo tampoco sería eficiente tener almacenados en disco sin orden todos estos valores, ya que habría que recorrer todos ellos hasta encontrar aquel que estuviésemos buscando, lo que, si se manejan unos volúmenes de información moderadamente grandes resultaría inviable, y por lo tanto el sistema sería inútil.

Por este motivo se necesita una clase que agrupe todos estos valores almacenados con una estructura bien definida que facilite el acceso y la actualización de los mismos.

Para ello se ha creado la clase `PeerConfidenceTable` (ver Figura 44) que, como su nombre indica, se basa en un conjunto de tablas en las que se agrupan los valores de reputación en *peers* según el grupo de intereses al que pertenezca, es decir, existe por ejemplo un valor de reputación para el *peer* "x" por los contenidos que nos ha facilitado dentro del grupo "y", y también estarán agrupados en función de la palabra clave en la que se esté evaluando dicho *peer*.

Veremos en este punto que las clases correspondientes al modelado de tablas de reputación en nuestro sistema, esto es, `ContentConfidenceTable`, `PeerConfidenceTable`, `PeerConfidenceTableGI` y `RiskTable` (las dos últimas se verán más adelante), tienen una estructura muy similar, presentando atributos muy parecidos entre ellas y con métodos de búsqueda y actualización cuya definición también es parecida.

Así, el atributo principal de esta clase es la tabla de reputación de *peers* propiamente dicha, implementada como una *Hashtable* que permitirá organizar la información tal y como se expuso en la descripción del sistema. Mediante este modelo, se podrá agrupar dichas reputaciones en función del grupo al que hacen referencia y de la palabra clave que tienen asociada.

En este caso también se cuenta con cuatro tablas auxiliares que permitirán conocer nombres e identificadores tanto de *peers* como de grupos, lo que hará que las búsquedas sean más eficientes y que la presentación de los datos sea más amigable. Para ello, esta clase cuenta con los métodos complementarios que permitirán acceder a los datos almacenados en estas tablas.

Por último, para terminar de definir esta clase, se debe hacer mención a los atributos que permiten localizar la tabla en disco, es decir, la ruta al directorio personal del usuario en formato de cadena de caracteres, que se podrá modificar en cualquier momento si así se desea.

Como ocurría antes, dos de los métodos más importantes de la clase son los correspondientes a la lectura y el almacenamiento de la tabla en disco: `toXMLFile` y `fromXMLFile` que serán claves para la persistencia de los datos.

Figura 44: Diagrama UML de la clase PeerConfidenceTable

Por otro lado, en lo que a funcionalidad del sistema se refiere, el cometido principal de esta clase consiste en buscar valores de reputación en *peer* dentro de la tabla correspondiente, mecanismo que implementa el método `findPeerConfidence`. Este método, permitirá conocer todos los *peers* que se ajusten a un patrón de búsqueda dado, como un listado de elementos `PeerConfidence`.

Del mismo modo, esta clase también se encarga de actualizar los valores de reputación de una de las entradas de la tabla en base al último intercambio de información llevado a cabo con el nodo correspondiente. El método encargado de esta tarea es `updatePeerConfidence`, y se encarga de buscar la entrada correspondiente y actualizarla.

4.3.5. PeerConfidenceTableGI

Acabamos de ver una de las clases más importantes del sistema, encargada de la organización de las reputaciones de los *peers* remotos en función de los grupos a los que pertenecen y a las palabras clave a las que hacen referencia. Esto nos da una fuente de información completa acerca de estos valores de reputación, pero en ocasiones esto va a suponer un grado de precisión demasiado alto a la hora de realizar consultas.

Por este motivo surge la necesidad de `PeerConfidenceTableGI`, una clase que permite tener un resumen de los valores de reputación en usuarios remotos independientemente del grupo en el que se haya interactuado con ellos. Sabemos que en una red *P2P* las comunicaciones entre nodos son continuas y en el caso de nuestro sistema pueden ir dirigidas a grupos de intereses concretos, pudiendo generar de esta forma multitud de entradas en nuestras tablas de confianza para un mismo *peer*.

Esta clase realiza un resumen de estas reputaciones dejando en un segundo plano a los grupos *JXTA* en los que se realizan los intercambios y centrándose en cambio en las palabras clave a las que hacen referencia las anotaciones. De esta forma obtendremos una medida más precisa de la calidad de las anotaciones que puede ofrecer un *peer* en una materia determinada. Esta circunstancia hace que la clase sea muy sencilla si la comparamos con `PeerConfidenceTable`, el volumen de datos que manejará será menor, y por lo tanto las búsquedas serán más rápidas.

En este caso, el atributo principal será una *Hashtable* similar a la de `PeerConfidenceTable` que no contemple división de la información por grupo de intereses, lo que supondrá una simplificación de la misma y una mayor eficiencia en las búsquedas (ver Figura 45).

Esta clase también cuenta con los atributos secundarios correspondientes a la ruta del fichero correspondiente en disco, como una cadena de caracteres, así como las tablas auxiliares que permitirán relacionar nombres e identificadores de *peers*.

Figura 45: Diagrama UML de la clase PeerConfidenceTableGI

Como ocurría con la clase anterior, se puede decir que la función más significativa de esta clase consiste en la búsqueda de información dentro de la tabla de reputación. Para este cometido, se ha implementado el método `findPeerConfidence`, que se encargará de buscar todas las entradas de la tabla que se ajusten a un patrón de búsqueda pasado como parámetro en forma de `PeerConfidence`. El resultado del método será un listado con todas las entradas encontradas.

La segunda funcionalidad más importante de esta clase consiste en actualizar el valor de reputación de los *peers* almacenados, mediante el método `updatePeerConfidence`. En primer lugar se deberá buscar la entrada a actualizar como un `PeerConfidence` y, posteriormente, invocar al método de actualización correspondiente.

Por último, también se debe hacer mención a los métodos encargados de la lectura del fichero correspondiente en disco y el almacenamiento de los datos en el mismo, funcionalidad que será fundamental de cara a conseguir la persistencia de los datos en el tiempo.

4.3.6. Risk

Con esta clase se completa la terna básica del sistema de gestión de la reputación. Aunque se hayan visto medidas de cálculo de la reputación de los *peers* (`PeerConfidence`) basadas en las correspondientes interacciones con contenidos (`ContentConfidence`), se necesita una última clase que nos ofrezca una medida objetiva que nos permita decidir si pedir una cierta información a un *peer* determinado o a otro.

La clase `Risk` modela un valor objetivo del riesgo que conlleva la interacción con un *peer* determinado, es decir, el factor riesgo de dicha comunicación. En este caso la medida se hace a través los tres parámetros vistos en la descripción del sistema *Poblano*, es decir, integridad, accesibilidad y rendimiento del *peer* al que hace referencia la entrada.

Para identificar unívocamente el nodo al que hace referencia cada una de las instancias de esta clase, se debe recoger tanto el nombre como el identificador único *JXTA* correspondientes, lo que garantizará que no haya duplicidades.

Como se puede observar en la figura 46, no hay ningún atributo de la clase que se corresponda con el valor del riesgo propiamente dicho, sino que será una combinación de los tres atributos definidos anteriormente, como se verá en el apartado de desarrollos algorítmicos.

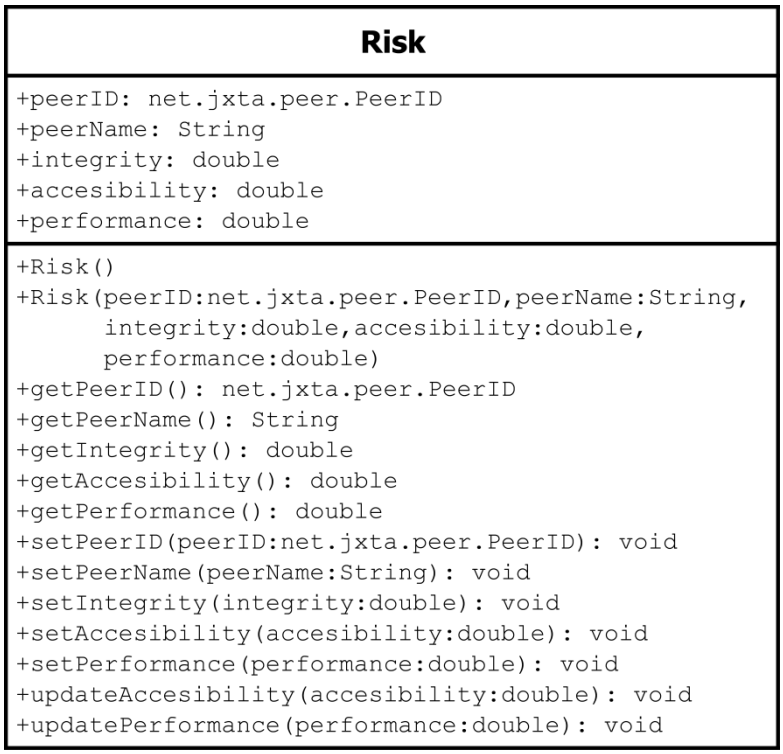


Figura 46: Diagrama UML de la clase RIsk

Una vez vistos los atributos de la clase se puede pasar a ver los métodos que los manejan. Al ser una clase relativamente sencilla, el grueso de los métodos desarrollados se basa en aquellos que sirven para establecer y consultar cada uno de los atributos, coloquialmente métodos `get` y `set`.

Como se ha mostrado anteriormente, cada vez que un *peer* accede a un contenido y lo califica, también se debe actualizar el valor de reputación del *peer* con el que se ha realizado la interacción, pero también debe ser modificado el riesgo que entraña la colaboración con dicho *peer* en base a su última acción. Por este motivo se definen dos nuevos métodos que se encargarán de actualizar estos valores.

4.3.7. RiskTable

Al igual que en el caso de `PeerConfidence` y `ContentConfidence`, la clase `Risk` es muy útil dentro del sistema de gestión de la reputación, pero no sirve de nada si no hay ningún orden definido que permita almacenar todos los valores disponibles y buscar entre ellos de forma eficiente. Por este motivo surge la necesidad de la clase `RiskTable` que, como su nombre indica, modela una tabla organizada de objetos de la clase `Risk`.

La organización de esta tabla, a diferencia de las tablas que recogen los valores de `ContentConfidence` y `PeerConfidence`, es mucho más sencilla, ya que no tiene en cuenta ni grupos ni palabras clave. Esto es debido a que por encima de todo, la clase `Risk` trata de reflejar el comportamiento global de un *peer*, evaluando

componentes de calidad que, no tienen una relación directa ni con grupos ni con intereses.

Cuando se quiere buscar un nuevo contenido y llegan al sistema respuestas de distintos *peers* se necesitará consultar siempre el factor de riesgo que conllevará el intentar la comunicación con cada uno de ellos. Para ordenar a este conjunto de *peers* el sistema se basará en los umbrales de cooperación, para cuyo cálculo será esencial la presencia del factor riesgo.

El atributo básico de esta clase, tal como muestra la figura 47 es la propia tabla de riesgos como una *Hashtable* que recogerá todas las entradas organizadas por el identificador del *peer* al que hacen referencia. Del mismo modo, y para completar la definición de la clase, se necesitarán atributos que determinen la ruta de dicha tabla, como una cadena de caracteres.

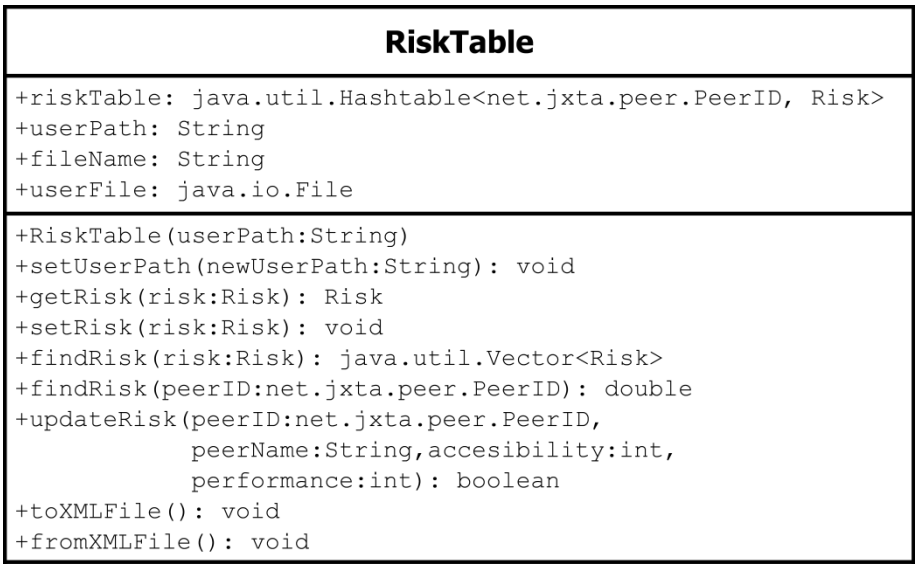


Figura 47: Diagrama UML de la clase RiskTable

Como ocurre con todas las clases que modelan las tablas del sistema, la funcionalidad principal de esta será la de buscar el riesgo que conlleva la interacción con un *peer* determinado, o con aquellos que se ajusten a un patrón dado.

De esta forma, también se encargará de actualizar los valores de riesgo correspondientes cada vez que obtenga nueva información al respecto, es decir, cada vez que se realice algún intercambio de información con el peer en cuestión.

Por último, esta clase también contará con los mecanismos de almacenaje y carga de datos que garanticen la persistencia de los datos, representados por los métodos `toXMLFile` y `fromXMLFile`.

4.3.8. TableManager

Una vez hecho el repaso por las clases que forman la base del funcionamiento del sistema de reputación, y se conoce mejor su constitución, habrá que dar un paso más para poder ver como se juntan todas las piezas anteriores en una estructura que las englobe a todos ellas. Este paso viene dado por la clase `TableManager`, que será la encargada de gestionar cualquier operación que se realice sobre las tablas anteriormente descritas, desde su carga, hasta su actualización, sin olvidar las búsquedas en las mismas.

La estructura de la clase es muy sencilla, ajustando el número de atributos al mínimo para tener toda la funcionalidad buscada y mantener la claridad de la misma.

Por otro lado, al ser una clase sobre la que recae todo el peso de la gestión de las tablas, el número de métodos que contiene es bastante elevado, ya que debe permitir el acceso a toda la información contenida en ellas, así como gestionar su interacción para que funcionen como un sistema plenamente operativo.

Esta clase, en definitiva, representa el núcleo del funcionamiento del sistema, y sería suficiente para implementar una aplicación que gestionase valores de reputación en el ámbito local, es decir, sin conexión alguna con otros nodos de la red. Esto no es útil para el cometido de este proyecto, pero sin duda alguna es una buena base para probar la funcionalidad de las tablas.

En cualquier caso, y como se ha hecho hasta ahora, se procederá con la explicación de los atributos y métodos contenidos en esta clase. Se podrá ver que, con la base adquirida al conocer todas las clases anteriores, es mucho más fácil entender la estructura de la misma.

En primer lugar se debe hacer especial mención a que esta clase se basa en la existencia de una instancia de cada una de las tablas vistas en los apartados anteriores: `ContentConfidenceTable`, `PeerConfidenceTableGI`, `RiskTable`, y `PeerConfidenceTable`, con el objetivo de centralizar toda la información y agrupar todos los mecanismos de búsqueda y actualización sobre ellas.

Como se ha dicho antes, el número de atributos de la clase no puede ser menor si queremos contar con toda la información necesaria para el funcionamiento del sistema de reputación. Existe un atributo por cada uno de los tipos de tabla que hemos visto hasta el momento, por lo que no necesitamos nada más para poder gestionar los valores de reputación.

La función de esta clase es, por tanto, exclusivamente la de gestionar toda la información del sistema de reputación, pero no solamente estos valores, sino también otros como por ejemplo información relativa a los nombres e identificadores de contenidos, *peers* y grupos.

Figura 48: Diagrama UML de la clase TableManager

Para realizar su trabajo, el método principal de esta clase es, sin duda, el constructor que, a partir de una cadena de caracteres que representa la ruta del directorio personal del usuario, se encarga de buscar los ficheros que contienen las tablas de reputación y cargarlas en memoria o, en caso de no existir, crea nuevas instancias desde cero para que vayan adquiriendo entradas a medida que el sistema intercambia información con otros nodos.

Si se consulta la figura 48, se puede observar que, al igual que ocurría con las tablas de reputación vistas en los apartados anteriores, los métodos más importantes de esta clase se dividen en dos bloques: el primero de ellos engloba todos aquellos métodos dedicados a buscar información dentro de las tablas, mientras que el otro gran grupo sirve para actualizar los valores de reputación almacenados en las mismas.

Una función interesante de esta clase consiste en proporcionar al sistema información sobre las palabras clave almacenadas en las tablas de contenidos, organizadas por grupos de intereses. Estos datos servirán para modificar el anuncio de *peer*, tal y como se explicó en el capítulo de descripción del sistema, y que será un mecanismo fundamental para la búsqueda de contenidos en la red.

`TableManager` también es el centro de datos que permite obtener y añadir información a las tablas auxiliares que se vieron en los apartados anteriores, de forma que se guarde un registro actualizado sobre los pares nombre-identificador de todos y cada uno de los contenidos, *peers* y grupos que se pueden encontrar en las tablas.

Como característica adicional, `TableManager` cuenta con el mecanismo que se encarga del guardado de las tablas, representado por el método `save`, que se invocará periódicamente durante la ejecución del sistema, y también en el cierre del mismo.

Por último, aunque no por ello menos importante, esta clase es la encargada de recabar toda la información necesaria de las tablas de reputación para obtener el umbral de cooperación correspondiente a un nodo dado y que, como se vio en la descripción del sistema, será fundamental a la hora de solicitar información a *peers* remotos.

4.3.9. Autosave

Al fin llegamos a la última de las clases utilizadas por el sistema de gestión de la reputación para implementar el módulo de gestión de tablas. Es una clase auxiliar, contenida dentro de la clase `Peer`, y que modela el funcionamiento de un hilo que se encarga del guardado periódico de las tablas locales almacenadas en la memoria.

La información de reputación y riesgo, contenida en las tablas locales, se actualiza constantemente cada vez que se busca un nuevo contenido, cada vez que se responde a una petición e incluso cuando se recibe el *feedback* de la puntuación de un *peer* remoto. Esto hace de la reputación y el riesgo datos altamente cambiantes a lo largo del tiempo.

Como se vio en la clase correspondiente al gestor de tablas (`TableManager`), el sistema cuenta con los mecanismos necesarios para el guardado de toda esta información, y dicho guardado se realiza al finalizar la ejecución del sistema. Esto proporciona la funcionalidad necesaria para que los datos sean persistentes entre sesiones y por lo tanto, hace que el sistema tenga memoria.

En cualquier caso, no se puede asegurar por completo que factores ajenos al sistema no provoquen un fallo en el mismo, bien sea una caída de la tensión o cualquier otra causa que fuerce una salida inesperada de la aplicación. En tal caso, si el sistema realizase el guardado únicamente al finalizar la sesión podría conllevar pérdidas importantes de datos, lo que sería inaceptable para el sistema. Por este motivo existe la clase `Autosave`, que lanza la ejecución de un hilo que realiza automáticamente guardados de las tablas cada cierto tiempo predeterminado.

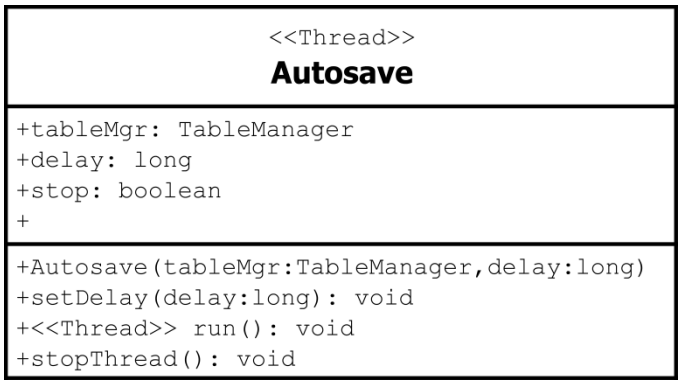


Figura 49: Diagrama UML de la case Autosave

Esta clase se compone básicamente de un atributo de tipo `TableManager` que permitirá acceder a los mecanismos de guardado de tablas cuando sea necesario, y también de un número que indica, en milisegundos, el tiempo que esperará el sistema desde que almacena la información actualizada de las tablas de reputación en disco hasta el siguiente guardado.

4.4. Módulo de gestión de logs

La gestión de *logs* es una tarea que, aunque no es tan visible como otras funciones del sistema, es de vital importancia para comprobar que todo funciona correctamente o, si no es así, qué es lo que está fallando.

Por este motivo, se necesita contar con un módulo específico que gestione todos los registros que se van generando, así como los ficheros en los que se guardarán estas entradas. Como se puede ver en la figura 50, este módulo está formado por una única clase, pero en el apartado siguiente veremos su utilidad.

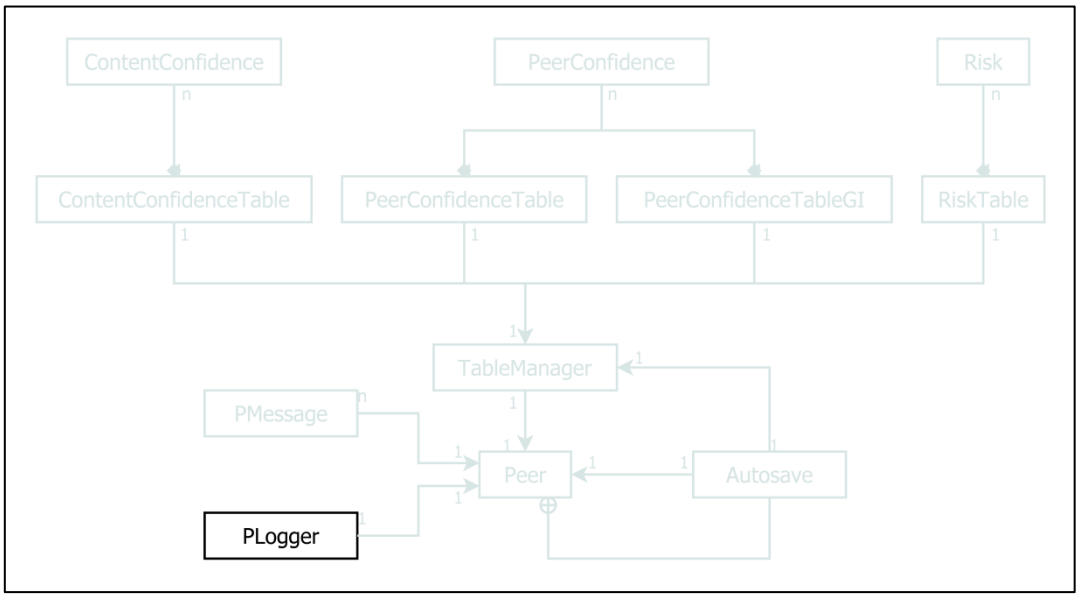


Figura 50: Módulo de gestión de logs

4.4.1. PLogger

`PLogger`, es el encargado de recoger un histórico con todas las acciones llevadas a cabo por el sistema de gestión de reputación.

La mayoría de las aplicaciones hoy en día, en paralelo a su funcionamiento, cuentan con un sistema de *log*, es decir, un mecanismo que se encarga de recoger y almacenar en archivos todos los eventos importantes que van sucediendo a lo largo de su ejecución, incluso los errores. Esta es una idea interesante que hemos decidido implementar en nuestro sistema de gestión de la reputación.

Para ello se ha creado una clase que gestiona ficheros de *log* en los que se recogen todas las interacciones realizadas con otros *peers*, tanto las iniciadas por el usuario local como aquellas originadas por otros usuarios de la red al enviarnos peticiones de información. El cometido de esta clase no solo se limita a la escritura de estos ficheros en el disco, sino que también se encargará de controlar la ubicación de los mismos e incluso el borrado de aquellos archivos que hayan quedado desactualizados.

La función de la clase que constituye este módulo de gestión de *logs* es bastante sencilla, ya que se limita exclusivamente a guardar los registros de la actividad en un fichero y permitir su posterior consulta.

Como punto adicional, esta clase también permite la gestión directa de estos ficheros con el objetivo de ahorrar espacio en disco. De esta forma, `PLogger` (ver Figura 51) incluirá los métodos necesarios para el borrado de ficheros de *log* antiguos.

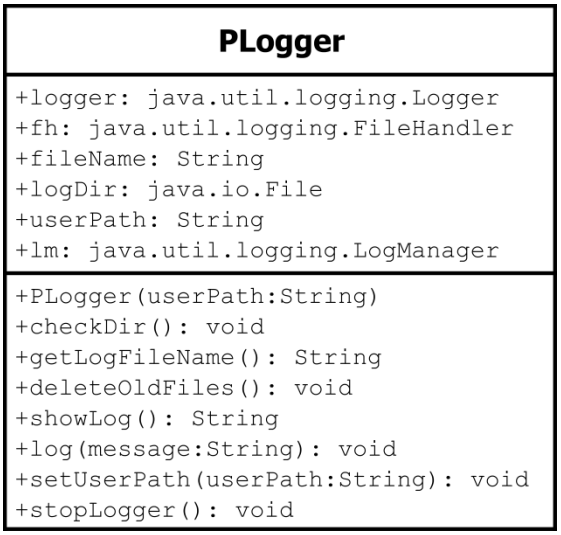


Figura 51: Diagrama UML de la clase PLogger

Al igual que ocurría en el caso de las tablas de reputación, la única información que necesita esta clase para comenzar su trabajo es una cadena de caracteres que represente la ruta al directorio personal del usuario. Una vez obtenida esta información, este módulo se encargará de la gestión de todos los ficheros que se creen a lo largo de la vida del sistema.

4.5. Módulo de gestión de la comunicación

Esta sección está dedicada a uno de los módulos más importantes, ya que sin él, el sistema estaría aislado de la red y no podría comunicarse con el resto de usuarios. Como se puede ver en la figura 52, está formado por una única clase, cuya importancia y utilidad se verá en el apartado siguiente.

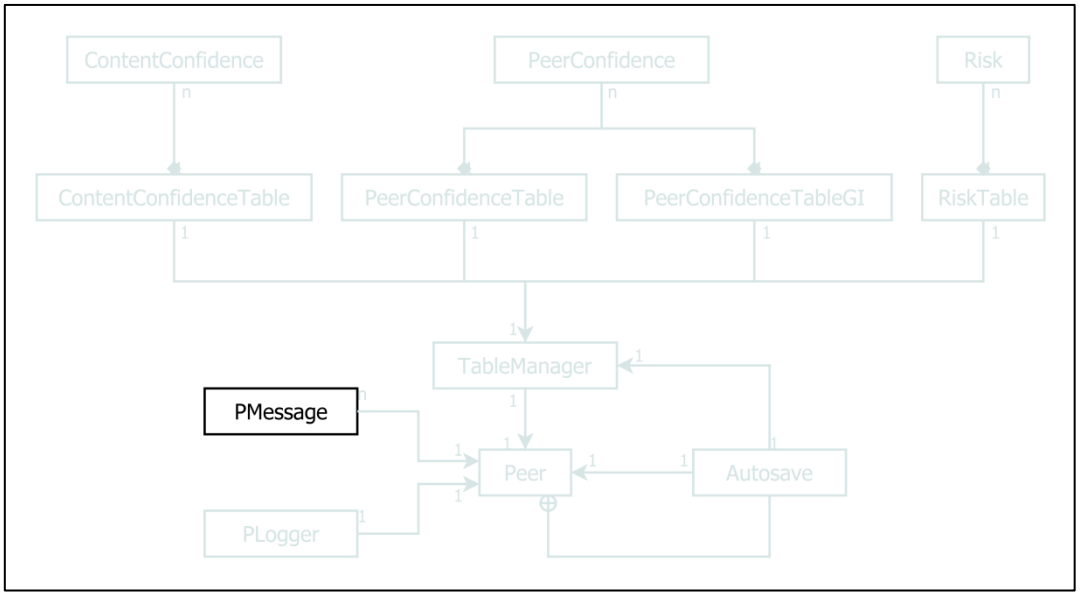


Figura 52: Módulo de gestión de la comunicación

4.5.1. PMessage

Por ahora se han visto clases que tienen relevancia dentro del sistema a nivel local, es decir, permiten que el sistema se ejecute sin interaccionar a priori con ningún otro *peer*. Por este motivo necesitamos dar un paso más y ser conscientes de que el sistema va a estar conectado a una red más grande, repleta de usuarios con los que poder comunicarse, y de los que podremos obtener información muy valiosa en lo que a reputación de contenidos y *peers* se refiere.

La estructura básica para el intercambio de información entre distintos *peers* dentro del sistema de gestión de la reputación es `PMessage`, clase que modela cualquier tipo de mensaje susceptible de ser enviado de un *peer* a otro, con el fin de recabar o proporcionar información.

Esta es una clase más compleja que las anteriores, ya que sobre ella recae el peso del intercambio de toda la información de cualquiera de las tablas locales presentes en el sistema. En este apartado se verá cómo funciona esta clase y qué atributos y métodos la componen, aunque se puede aprender más de ella en el apartado 3.1 del capítulo 3, donde se pueden ver todos los tipos de mensaje disponibles en el sistema de reputación.

Aunque pueda parecer una estructura sencilla de acuerdo con la figura 53, el elevado número de campos susceptibles de estar presentes en un mensaje del sistema, así como el gran número de mecanismos necesarios para el procesado de todos ellos,

hacen de esta clase una de las más complejas del sistema y, por lo tanto, merece una especial atención dentro de este proyecto.

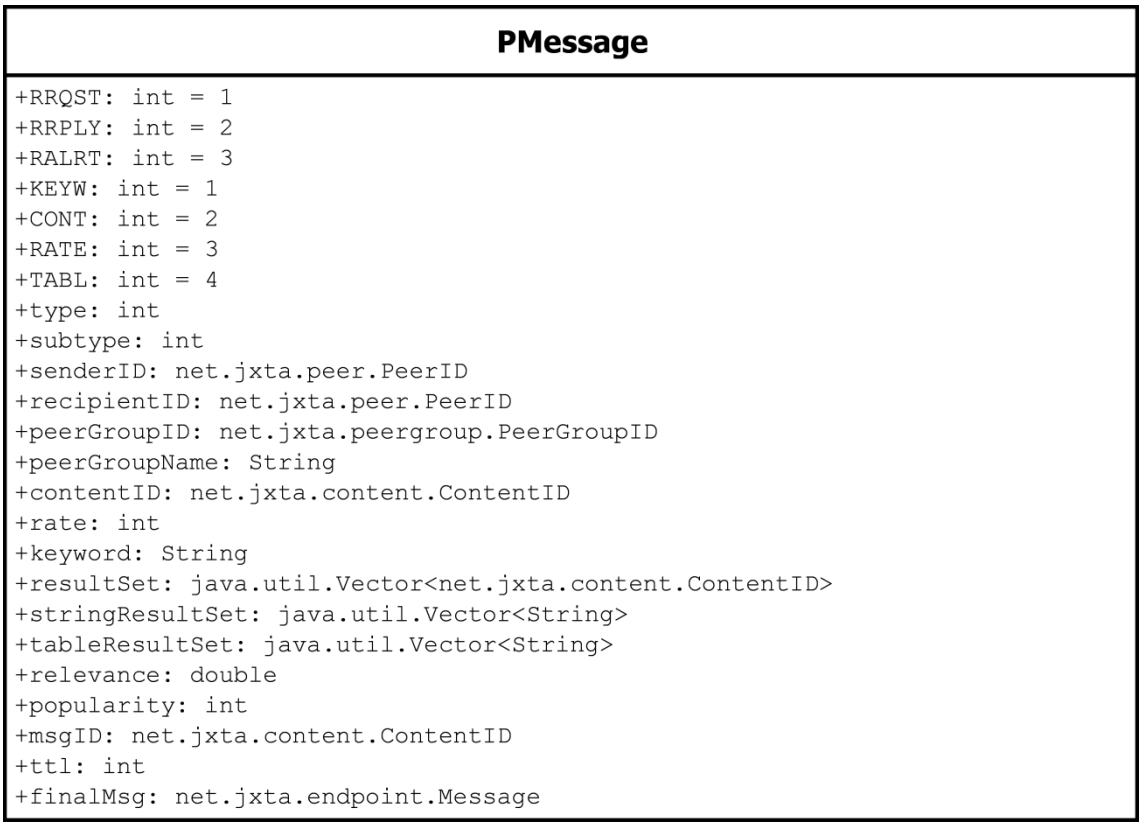


Figura 53: Diagrama UML de la clase PMessage: Atributos

En primer lugar, un mensaje debe contener información suficiente para el encaminado de la información. Estos campos contendrán los nombres e identificadores de emisor y receptor que permitan verificar si el encaminamiento ha sido correcto.

Por otro lado, los mensajes deben contener información acerca de su carga útil, esto es, campos que permitan identificar en el caso de las peticiones los datos que se solicitan, y en el caso de las respuestas, toda la información relativa a esos datos.

Dos campos de especial importancia en esta clase son: el identificador del mensaje, que será un identificador único *JXTA*, y que permitirá emparejar peticiones con sus correspondientes respuestas; y el tiempo de vida del mensaje, o *ttl*, que indicará el número de veces que se puede reenviar el mismo.

En lo que a contenido del mensaje respecta, se debe especificar que, para su manejo dentro del sistema, el mensaje tendrá un formato *XML* que facilitará su procesamiento. Por otro lado, cuando dicho mensaje se desee enviar por la red hasta su destinatario, este deberá convertirse a un formato determinado por la plataforma *JXTA*.

Una vez mencionados todos estos atributos, se puede observar la versatilidad de la clase y su utilidad dentro del sistema de gestión de la reputación, y por lo tanto se puede

suponer que los métodos que determinan la funcionalidad son de igual importancia para el sistema.

```
+PMessage()
+PMessage(msg:net.jxta.endpoint.Message)
+getMessage(): net.jxta.endpoint.Message
+toString(): String
+getLogString(): String
+parseStringRSE(): void
+getType(): int
+getSubtype(): int
+getSenderID(): net.jxta.peer.PeerID
+getSenderName(): String
+getRecipientID(): net.jxta.peer.PeerID
+getPeerGroupID(): net.jxta.peergroup.PeerGroupID
+getPeerGroupName(): String
+getContentID(): net.jxta.content.ContentID
+getContentName(): String
+getKeyword(): String
+getRate(): int
+getResultSet(): java.util.Vector<net.jxta.content.ContentID>
+getStringResultSet(): java.util.Vector<String>
+getTableResultSet(): java.util.Vector<String>
+getRelevance(): double
+getPopularity(): int
+getMessageID(): net.jxta.content.ContentID
+getTTL(): int
+setType(type:int): void
+setSubtype(subtype:int): void
+setSenderID(senderID:net.jxta.peer.PeerID): void
+setSenderName(senderName:String): void
+setRecipientID(recipientID:net.jxta.peer.PeerID): void
+setPeerGroupID(peerGroupID:net.jxta.peergroup.PeerGroupID): void
+setPeerGroupName(peerGroupName:String): void
+setContentID(contentID:net.jxta.content.ContentID): void
+setContentName(contentName:String): void
+setKeyword(keyword:String): void
+setRate(rate:int): void
+setResultSet(resultSet:java.util.Vector<net.jxta.content.ContentID>): void
+addElementToResultSet(contentID:net.jxta.content.ContentID): void
+setStringResultSet(stringResultSet:java.util.Vector<String>): void
+addElementToStringResultSet(stringEntry:String): void
+setTableResultSet(tableResultSet:java.util.Vector<String>): void
+addElementToTableResultSet(tableEntry:String): void
+setRelevance(relevance:double): void
+setPopularity(popularity:int): void
+setMessageID(msgID:net.jxta.content.ContentID): void
+setTTL(ttl:int): void
```

Figura 54: Diagrama UML de la clase PMessage: Métodos

Caben destacar los constructores de la clase, que permitirán generar un mensaje vacío al que se irán añadiendo los campos correspondientes o, en otro caso, también se podrá crear un mensaje directamente de la información recibida por la red desde un nodo remoto, facilitando así el procesamiento del mismo.

Este último caso es interesante ya que, cuando se recibe un mensaje de la red, hay ocasiones especialmente críticas en procesamiento de los datos que contiene, como es el

caso de la respuesta a una petición de tabla de reputación. En este caso, será crucial el correcto funcionamiento si se quieren recuperar los valores reales contenidos en la original.

Por último, se debe mencionar la utilidad de esta clase en el proceso de gestión de *logs*, ya que los mensajes recibidos de la red, también deberán ser contemplados en los ficheros que registran la actividad. Por este motivo, la clase `PMessage` deberá contar con los mecanismos necesarios para generar las entradas del fichero de *log* en un formato inteligible.

4.6. Módulo de gestión de la reputación

Por último se debe dedicar una sección al último módulo del sistema, el de gestión de la reputación, que será el encargado de coordinar a todos los módulos que se han visto hasta el momento y que, por lo tanto, será vital para el funcionamiento del conjunto.

Como se puede ver en la figura 55, este módulo estará formado por una única clase, pero, como se explicará en el apartado siguiente, la complejidad y la utilidad de la misma lo justifican.

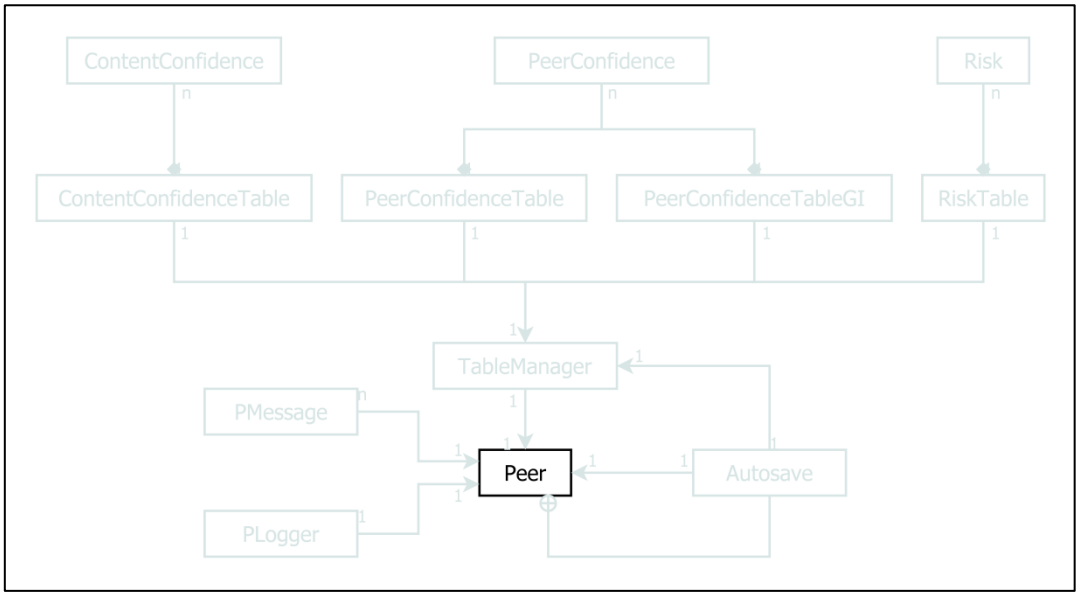


Figura 55: Módulo de gestión de la reputación

4.6.1. Peer

Para que todas las clases anteriores funcionen y se comporten en conjunto como un verdadero sistema se debe disponer de una clase de nivel superior que las englobe a todas ellas y que sea capaz de implementar la lógica necesaria para manejarlas. Esta clase se llamará `Peer` (ver Figura 56) y modelará al usuario que es capaz de conectarse a la red *JXTA*, manejar todas las tablas de confianza, y comunicarse con el resto de *peers* participantes en la red para intercambiar información.

Figura 56: Diagrama UML de la clase Peer. Atributos

Se puede establecer una relación directa entre cualquier instancia de la clase `Peer` y un *peer* de la red *JXTA*. De la misma forma también se puede identificar a dicho *peer* como un usuario que se conecta a la red *JXTA* voluntariamente y al que se le asigna un identificador único para diferenciarlo de los demás y que perdurará en el tiempo.

Cada usuario, como se ha visto hasta ahora, será independiente del resto de usuarios o *peers* de la red, incluso de aquellos que se conecten desde el dispositivo utilizado por este primero. Para esto hemos visto que todos, aún coexistiendo en el mismo dispositivo, tendrán espacios de trabajo distintos e identificadores diferentes, por lo que se podrán comunicar entre ellos, creando y evaluando nuevos contenidos para la pervivencia del sistema de reputación.

La clase *peer* se encargará de manejar todas las clases vistas hasta ahora, es decir, contará con un objeto de tipo `TableManager` para gestionar todas las tablas de confianza locales, tendrá un objeto `PLogger` que le permita dejar constancia de todas las actividades que realice durante el tiempo que esté conectado a la red y podrá crear mensajes de tipo `PMessage` para comunicarse con otros usuarios.

Esta clase también contendrá todos los atributos específicos de la red *JXTA* para garantizar su conectividad dentro de la red y dentro del sistema de reputación implementado, tal y como se representa en la figura 56.

Además de estos atributos, el sistema necesita toda la información necesaria para identificar al nodo dentro de la red. Por ello, deberá contar con una cadena de caracteres que representará el nombre del *peer*, un identificador *JXTA* único que lo diferenciará del resto de nodos de la red, y un listado con los grupos a los que pertenece.

Como información adicional para verificar que el sistema funciona tal y como el usuario ha determinado, esta clase contará con atributos que representen las opciones de configuración definidas en el fichero correspondiente, que se consultará al inicio de la ejecución del programa. Estos valores de configuración, como son el tiempo de vida de los mensajes o el directorio personal de usuario podrán modificarse a lo largo del ciclo de vida del sistema a través de los métodos correspondientes.

Como se ha visto, esta clase contiene un número considerable de atributos, cuyo cometido, en la mayoría de los casos, es adaptar este sistema de gestión de la reputación a la red *P2P JXTA*. Se verá a partir de ahora cómo los métodos contenidos en la misma van dirigidos por igual a establecer una relación directa entre ambas tecnologías, haciendo trabajar al unísono elementos de cada una de ellas con el fin de conseguir un sistema eficiente y robusto.

```
+Peer(name:String)
+stringToIDGroup(stringGroupID:String): net.jxta.peergroup.PeerGroupID
+getAvailableContents(targetGID:net.jxta.peergroup.PeerGroupID,
                     keyword:String): java.util.Vector<net.jxta.content.ContentID>
+<<ContentIface>> removeContentFromGroup(targetGID:net.jxta.peergroup.PeerGroupID,
                                         keyword:String,
                                         targetCID:net.jxta.content.ContentID): boolean
+<<ContentIface>> addNewContent(peerGroupID:net.jxta.peergroup.PeerGroupID,
                                contentName:String,
                                keyword:String): net.jxta.content.ContentID
+<<ContentIface>> addKeywordToContent(contentID:net.jxta.content.ContentID,
                                      keyword:String): void
+<<PeerIface>> addPeersToTable(peerGroupID:net.jxta.peergroup.PeerGroupID): boolean
+<<GroupIface>> createNewGroup(groupName:String,
                               description:String): net.jxta.peergroup.PeerGroupID
+<<GroupIface>> findNewGroup(): java.util.Vector<net.jxta.peergroup.PeerGroupID>
+sendTableRequest(destPeerTable:java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                  net.jxta.peer.PeerID>): boolean
+sendContentRequest(destPeerTable:java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                    java.util.Vector<net.jxta.peer.PeerID>>,
                    keyword:String): boolean
+sendKeywordRequest(destPeerList:java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                    java.util.Vector<net.jxta.peer.PeerID>>,
                    keyword:String): boolean
+sendRateRequest(keyword:String,targetPID:net.jxta.peer.PeerID,
                targetGID:net.jxta.peergroup.PeerGroupID,
                targetCID:net.jxta.content.ContentID,
                relevance:double,popularity:int): boolean
+findInContentTable(keyword:String,gID:net.jxta.peergroup.PeerGroupID,
                   inGroup:boolean): java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                    java.util.Vector<net.jxta.content.ContentID>>
+findInPeerTable(keyword:String,gID:net.jxta.peergroup.PeerGroupID,
                inGroup:boolean): java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                 java.util.Hashtable<net.jxta.peer.PeerID,
                 java.util.Vector<net.jxta.content.ContentID>>>
+findInAdvertisements(keyword:String,gID:net.jxta.peergroup.PeerGroupID,
                     inGroup:boolean): java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                      java.util.Hashtable<net.jxta.peer.PeerID,
                      java.util.Vector<net.jxta.content.ContentID>>>
+<<ContentIface>> findContent(keyword:String,
                              gID:net.jxta.peergroup.PeerGroupID,
                              forceRemote:boolean): java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                               java.util.Hashtable<net.jxta.peer.PeerID,
                               java.util.Vector<net.jxta.content.ContentID>>>
+selectPeer(peerTable:java.util.Hashtable<net.jxta.peer.PeerID,
            java.lang.Double>): net.jxta.peer.PeerID
+<<ContentIface>> rateContent(keyword:String,
                              targetGID:net.jxta.peergroup.PeerGroupID,
                              foundGID:net.jxta.peergroup.PeerGroupID,
                              targetCID:net.jxta.content.ContentID,
                              targetPID:net.jxta.peer.PeerID,
                              rating:double): void
+stopPeer(): void
+<<net.jxta.endpoint.EndpointListener>> processIncommingMessage(msg:net.jxta.endpoint.Message,
                                                                src:net.jxta.endpoint.EndpointAddress,
                                                                dst:net.jxta.endpoint.EndpointAddress): void
+updatePeerRisk(): void
+getAvailableGroups(): java.util.Vector<String>
+insertAdvDescription(doc3:org.w3c.dom.Document): void
+<<PeerIface>> getPeerConfidenceValue(pid:net.jxta.peer.PeerID,
                                      gid:net.jxta.peergroup.PeerGroupID,
                                      keyword:String): double
+<<ContentIface>> getContentConfidenceValue(cid:net.jxta.content.ContentID,
                                            gid:net.jxta.peergroup.PeerGroupID,
                                            keyword:String): double
+documentToString(doc:org.w3c.dom.Document): String
+containsKeyword(peerAdv:net.jxta.protocol.PeerAdvertisement,
                 keyword:String,gID:net.jxta.peergroup.PeerGroupID): boolean
+identifyGroups(peerAdv:net.jxta.protocol.PeerAdvertisement,
                keyword:String): java.util.Vector<net.jxta.peergroup.PeerGroupID>
+descToDocument(sd:net.jxta.document.StructuredDocument): org.w3c.dom.Document
+getKeywordFromDocument(doc:org.w3c.dom.Document): java.util.Hashtable<net.jxta.peergroup.PeerGroupID,
                        java.util.Vector<String>>
+obtainPort(): int
+isAvailablePort(tcpPort:int): boolean
+loadAvailableGroups(): void
+loadProperties(): void
+<<AdminIface>> saveProperty(name:String,
                             value:String,
                             now:boolean): void
+<<AdminIface>> listProperties(): String
+<<AdminIface>> showLog(): String
+<<AdminIface>> deleteOldLogFiles(): void
+getContentName(contentID:net.jxta.content.ContentID): String
+getContentID(contentName:String): net.jxta.content.ContentID
+getPeerName(peerID:net.jxta.peer.PeerID): String
+getPeerID(peerName:String): net.jxta.peer.PeerID
+getGroupName(peerGroupID:net.jxta.peergroup.PeerGroupID): String
+getGroupID(peerGroupName:String): net.jxta.peergroup.PeerGroupID
+republishAdvertisement(): void
+flushLocalAdvertisements(type:int): void
```

Figura 57: Diagrama UML de la clase Peer. Métodos

Tal y como se puede observar en el diagrama *UML* de la figura 57, la clase `Peer` implementa varios interfaces que serán descritos en el apartado correspondiente, y por lo tanto heredarán los métodos definidos en cada uno de ellos. En esta descripción de los métodos haremos un repaso por todos ellos, explicando los particulares de esta clase e indicando los que se hereden de los interfaces.

La funcionalidad principal de esta clase de cara a su conectividad consiste en la posibilidad de generar y enviar mensajes de petición de cualquiera de los tipos definidos en el capítulo 3, utilizando el módulo de comunicación para su creación y aplicando las funciones de *JXTA* para su transmisión.

Siguiendo en la línea de la comunicación, esta clase proporciona la funcionalidad de recibir y procesar cualquier tipo de mensaje que se reciba de la red, ya sean peticiones o respuestas. Esto sucede de forma transparente gracias a los mecanismos proporcionados por *JXTA*, permitiendo a la aplicación trabajar con el sistema sin verse afectada por los mensajes entrantes.

En el ámbito local, ya que posee un atributo de tipo `TableManager`, que le otorga una total conectividad con el módulo de gestión de tablas, esta clase contará con todos los mecanismos para realizar búsquedas de contenidos en las tablas locales, así como en los anuncios de red propagados por otros nodos.

CAPÍTULO 5: PRUEBAS DEL SISTEMA

5.1. Introducción

Tras la primera implementación de un sistema como el que se ha descrito en los capítulos anteriores no se puede dar por hecho que todo funcionará a la perfección desde el primer momento, aunque eso sería lo ideal. La realidad nos muestra que el desarrollo de una aplicación pasa por la comprobación de que ésta cumple con todos los requisitos para los que ha sido diseñada, al igual que cumple con toda la funcionalidad prevista.

Para conseguir esto se necesitará que cada elemento del sistema pase por una batería de pruebas que permita comprobar si la implementación llevada a cabo es correcta, evitando así posibles errores que puedan aparecer en el futuro. Cada una de estas pruebas se ha realizado aislando el módulo en cuestión e implementando una pequeña aplicación para comprobar su funcionamiento.

A lo largo de este capítulo se describen todas las pruebas a las que ha sido sometido cada módulo del sistema, presentando a través de tablas un informe de resultados en el que se mostrará si el sistema cumple con las expectativas. El capítulo estará dividido en las siguientes secciones:

- Pruebas de estructuras básicas.
- Pruebas de tablas.
- Pruebas de *log*.
- Pruebas de comunicación.
- Pruebas del sistema completo.
- Pruebas de la aplicación de prueba.
- Pruebas de robustez.

5.2. Pruebas de estructuras básicas

5.2.1. Confianza en contenido

La primera de las estructuras que se abordarán en esta sección será la que modela los valores de confianza en contenidos. Por ser una de las bases en las que se sustenta el sistema de gestión de reputación, las pruebas realizadas han sido exhaustivas, siendo los resultados los siguientes:

Descripción de la Prueba	Resultado
Creación de un valor de confianza en contenido por defecto	Correcto
Creación de un valor de confianza en contenido pasando sus componentes como parámetro	Correcto
Obtención de los valores almacenados en los	Correcto

componentes de la confianza	
Modificación de los valores almacenados en los componentes de la confianza	Correcto
Incremento del valor de popularidad	Correcto
Actualización del valor de confianza por calificación local	Correcto
Actualización del valor de confianza por *feedback* remoto	Correcto

Tabla 1: Pruebas de confianza en contenido

5.2.2. *Confianza en peer*

La segunda de las estructuras básicas para el modelado de los valores de confianza, es decir, la que recoge la confianza en *peers*, también debe ser objeto de un conjunto de pruebas que verifique su funcionamiento de forma minuciosa. Las pruebas llevadas a cabo y sus resultados pueden verse en la siguiente tabla:

Descripción de la Prueba	Resultado
Creación de un valor de confianza en *peer* por defecto	Correcto
Creación de un valor de confianza en *peer* pasando sus componentes como parámetro	Correcto
Obtención de los valores almacenados en los componentes de la confianza	Correcto
Modificación de los valores almacenados en los componentes de la confianza	Correcto
Incremento del valor de popularidad	Correcto
Actualización del valor de confianza	Correcto

Tabla 2: Pruebas de confianza en *peer*

5.2.3. *Factor de riesgo*

El tercero de los elementos básicos del sistema de gestión de reputación, como se mostró en el capítulo de descripción del sistema, será el factor de riesgo que conlleva la interacción con otro nodo de la red. Esto implica que el correcto funcionamiento de esta estructura es de vital importancia para el funcionamiento del sistema, por lo que la batería de pruebas será la siguiente:

Descripción de la Prueba	Resultado
Creación de un valor de riesgo por defecto	Correcto
Creación de un valor de riesgo pasando sus componentes como parámetro	Correcto
Obtención de los valores almacenados en los componentes del riesgo	Correcto
Actualización del parámetro de accesibilidad	Correcto
Actualización del parámetro de rendimiento	Correcto

Tabla 3: Pruebas de factor de riesgo

5.2.4. Mensajes

Para terminar con las estructuras fundamentales del sistema no se debe olvidar el mensaje, que será la base para la intercomunicación de los nodos. Por su importancia dentro del sistema, ésta debe ser una de las estructuras a las que se debe prestar especial atención, ya que un incorrecto manejo de los mensajes puede ser causa del aislamiento de un *peer*. Las pruebas a las que se ha sometido esta estructura son las siguientes:

Descripción de la Prueba	Resultado
Creación de un mensaje del sistema por defecto	Correcto
Creación de un mensaje del sistema a partir de un mensaje *JXTA*	Correcto
Construcción de un mensaje *JXTA* a partir de los campos del mensaje	Correcto
Creación de una entrada de *log* a partir de un mensaje del sistema	Correcto
Obtención de los valores almacenados en los campos del mensaje	Correcto
Modificación de los valores almacenados en los campos del mensaje	Correcto

Tabla 4: Pruebas de mensaje

5.3. Pruebas de tablas

5.3.1. Tabla de confianza en contenidos

Las pruebas correspondientes a los valores de confianza en contenidos resultaron satisfactorias, tal y como se ha podido comprobar en la sección anterior, por lo que será el momento de presentar las pruebas de la tabla que engloba todos estos valores. Son las siguientes:

Descripción de la Prueba	Resultado
Carga de la tabla en memoria a partir de la ruta del directorio de usuario	Correcto
Cambio de la ubicación del directorio de usuario	Correcto
Comprobación de la existencia de una confianza en contenido dado su identificador y el grupo al que pertenece	Correcto
Obtención de un listado de los contenidos que se ajustan a un patrón de búsqueda dado	Correcto
Búsqueda de un valor de confianza en contenido concreto	Correcto
Adición de un valor de confianza a la tabla	Correcto
Actualización de un valor de confianza por calificación local	Correcto
Actualización de un valor de confianza por *feedback* remoto	Correcto
Obtención de un documento *XML* con las palabras clave y grupos de la tabla para su publicación en el anuncio de *peer*	Correcto
Eliminación de un contenido concreto de la tabla de confianza	Correcto
Obtención del nombre de un contenido dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un contenido dado su nombre	Correcto
Obtención del nombre de un grupo dado su identificador *JXTA* único	Correcto

Descripción de la Prueba	Resultado
Obtención del identificador *JXTA* único de un grupo dado su nombre	Correcto
Almacenamiento de la tabla completa en disco	Correcto
Lectura de la tabla completa de disco	Correcto

Tabla 5: Pruebas de tabla de confianza en contenidos

5.3.2. Tabla de confianza en peers dependiente de grupo

Al igual que se hizo con los contenidos, se debe evaluar la funcionalidad de la tabla que recoge los valores de confianza en *peers*, dependiendo del grupo al que estos pertenezcan. La batería de pruebas en este caso ha sido la siguiente:

Descripción de la Prueba	Resultado
Carga de la tabla en memoria a partir de la ruta del directorio de usuario	Correcto
Cambio de la ruta que define la ubicación del directorio de usuario	Correcto
Actualización de un valor de confianza en *peer* debido a una calificación local o un *feedback*	Correcto
Obtención de un listado de confianzas en *peer* que se ajustan a un patrón de búsqueda dado	Correcto
Búsqueda de un valor de confianza en *peer* concreto	Correcto
Modificación de un valor de confianza en *peer* concreto	Correcto
Obtención del nombre de un *peer* dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un *peer* dado su nombre	Correcto
Obtención del nombre de un grupo dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un grupo dado su nombre	Correcto
Almacenamiento de la tabla completa en disco	Correcto

Descripción de la Prueba	Resultado
Lectura de la tabla completa de disco	Correcto

Tabla 6: Pruebas de tabla de confianza en *peer* dependiente de grupo

5.3.3. Tabla de confianza en peers independiente de grupo

Los valores de confianza en *peer*, tal y como se presentó en el capítulo de descripción general del sistema, se pueden organizar en tablas independientes de grupo. Por ello, la funcionalidad de dicha tabla será susceptible de ser verificada mediante su correspondiente batería de pruebas. Han sido las siguientes:

Descripción de la Prueba	Resultado
Carga de la tabla en memoria a partir de la ruta del directorio de usuario	Correcto
Cambio de la ruta que define la ubicación del directorio de usuario	Correcto
Actualización de un valor de confianza en *peer* debido a una calificación local o un *feedback*	Correcto
Obtención de un listado de confianzas en *peer* que se ajustan a un patrón de búsqueda dado	Correcto
Búsqueda de un valor de confianza en *peer* concreto	Correcto
Modificación de un valor de confianza en *peer* concreto	Correcto
Obtención del nombre de un *peer* dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un *peer* dado su nombre	Correcto
Almacenamiento de la tabla completa en disco	Correcto
Lectura de la tabla completa de disco	Correcto

Tabla 7: Pruebas de tabla de confianza en *peer* independiente de grupo

5.3.4. Tabla de riesgos

Para completar las pruebas de las tablas que conforman el sistema de gestión de la reputación se debe hacer referencia a aquellas realizadas sobre la tabla que recoge todos los valores de riesgo almacenados en el sistema. Son las siguientes:

Descripción de la Prueba	Resultado
Carga de la tabla en memoria a partir la ruta del directorio de usuario	Correcto
Cambio de la ruta que define la ubicación del directorio de usuario	Correcto
Actualización de un valor de riesgo debido a una calificación local o un *feedback*	Correcto
Obtención de un listado de riesgos que se ajustan a un patrón de búsqueda dado	Correcto
Búsqueda de un valor de riesgo concreto	Correcto
Modificación de un valor riesgo concreto	Correcto
Almacenamiento de la tabla completa en disco	Correcto
Lectura de la tabla completa de disco	Correcto

Tabla 8: Pruebas de tabla de riesgos

5.3.5. Gestor de tablas

Por último, y a pesar de que el funcionamiento de cada una de las tablas por separado haya resultado satisfactorio, se debe realizar la comprobación del módulo que engloba el manejo de todas ellas, es decir, el gestor de tablas del sistema. De nuevo en este caso no se va a dar por hecho que el funcionamiento de sus componentes signifique el correcto funcionamiento del módulo completo, por lo que las pruebas realizadas son las siguientes:

Descripción de la Prueba	Resultado
Carga de las tablas en memoria dada la ruta del directorio de usuario	Correcto
Cambio de la ruta del directorio de usuario	Correcto
Obtención del documento estructurado que se insertará en el anuncio de *peer JXTA*	Correcto
Obtención del listado de grupos disponible	Correcto
Actualización de un valor de confianza debido a una calificación local	Correcto
Actualización de un valor de confianza debido a un *feedback* remoto	Correcto

Descripción de la Prueba	Resultado
Obtención de un listado de contenidos locales que se ajustan a un patrón de búsqueda dado	Correcto
Obtención de un listado de *peers* relacionados con un grupo y una palabra clave concretos	Correcto
Obtención de un valor de confianza en contenido concreto de la tabla local	Correcto
Obtención de un valor de confianza en *peer* concreto de la tabla local	Correcto
Obtención de un valor de confianza en *peer* concreto en la tabla independiente de grupo	Correcto
Obtención de un listado de confianza en *peer* para un grupo concreto	Correcto
Actualización de la tabla de confianza en *peer* de un grupo concreto	Correcto
Eliminación de un valor de confianza en contenido concreto	Correcto
Cálculo del umbral de cooperación de un *peer* remoto	Correcto
Actualización de un valor de riesgo de la tabla local	Correcto
Obtención del nombre de un contenido dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un contenido dado su nombre	Correcto
Obtención del nombre de un *peer* dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un *peer* dado su nombre	Correcto
Obtención del nombre de un grupo dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un grupo dado su nombre	Correcto
Salvado de las tablas en disco	Correcto

Descripción de la Prueba	Resultado
Detención del gestor de tablas completo	Correcto

Tabla 9: Pruebas del gestor de tablas

5.4. Pruebas de log

5.4.1. Gestor de logs

En las secciones anteriores se presentaron las pruebas realizadas sobre todas las estructuras fundamentales para la gestión de los valores de confianza propiamente dichos. Como se explicó en la implementación del sistema existen también módulos auxiliares, como el correspondiente a la gestión de *logs* del sistema, que deben someterse a pruebas para verificar su funcionamiento. Las pruebas en este caso y los resultados obtenidos son los siguientes:

Descripción de la Prueba	Resultado
Creación del fichero de *log* en base a la ruta del directorio de usuario	Correcto
Comprobación de la carpeta contenedora de los ficheros de *log*	Correcto
Creación del nombre del fichero de *log* actual	Correcto
Obtención del contenido del fichero de *log* actual	Correcto
Inserción de una nueva entrada en el fichero de *log*	Correcto
Parada del sistema gestor de *logs*	Correcto

Tabla 10: Pruebas del gestor de logs

5.5. Pruebas de comunicación

Antes de continuar con la enumeración de las pruebas realizadas sobre otros sistemas se debe incidir en la comprobación de los mecanismos de comunicación disponibles para el intercambio de información entre los nodos de la red. Entre estas funciones se encontrarán la creación de los mensajes correspondientes, y sus pruebas han sido las siguientes:

Descripción de la Prueba	Resultado
Creación y envío de petición de búsqueda de anotaciones	Correcto
Creación y envío de petición de propagación de	Correcto

Descripción de la Prueba	Resultado
anotaciones	
Creación y envío de petición de evaluación de anotaciones	Correcto
Creación y envío de petición de intercambio de tabla de confianza en *peer*	Correcto
Creación y envío de respuesta de búsqueda de anotaciones	Correcto
Creación y envío de respuesta de propagación de anotaciones	Correcto
Creación y envío de respuesta de evaluación de anotaciones	Correcto
Creación y envío de respuesta de intercambio de tabla de confianza en *peer*	Correcto
Recepción y procesamiento de petición de búsqueda de anotaciones	Correcto
Publicación del anuncio de *peer*	Correcto
Recepción y procesamiento de petición de propagación de anotaciones	Correcto
Recepción y procesamiento de petición de evaluación de anotaciones	Correcto
Recepción y procesamiento de petición de intercambio de tabla de confianza en *peer*	Correcto
Recepción y procesamiento de respuesta de búsqueda de anotaciones	Correcto
Recepción y procesamiento de respuesta de propagación de anotaciones	Correcto
Recepción y procesamiento de respuesta de evaluación de anotaciones	Correcto
Recepción y procesamiento de respuesta de intercambio de tabla de confianza en *peer*	Correcto
Obtención y procesamiento de anuncios de *peer* de la red	Correcto

Descripción de la Prueba	Resultado
Obtención y procesamiento de anuncios de grupo de la red	Correcto
Actualización del anuncio de *peer* y nueva publicación del mismo	Correcto

Tabla 11: Pruebas de comunicación

5.6. Pruebas del sistema completo

Como ha ocurrido hasta ahora, a pesar del correcto funcionamiento de los componentes individuales del sistema, se deben comprobar todos ellos como conjunto. Para esto se pasará una batería de pruebas al módulo principal del sistema, que será el encargado de que todos los componentes actúen al unísono y se comporten como una única entidad. Las pruebas realizadas son las siguientes:

Descripción de la Prueba	Resultado
Inicialización del sistema dado el nombre de usuario	Correcto
Obtención de un listado de contenidos tras la búsqueda por grupo y palabra clave	Correcto
Eliminación de un valor de confianza en contenido de la tabla local	Correcto
Creación de un nuevo contenido en las tabla local de confianza	Correcto
Etiquetado de un contenido con una nueva palabra clave	Correcto
Actualización de las tablas de confianza en *peer* con nuevos resultados	Correcto
Creación de un nuevo grupo de intereses	Correcto
Obtención de un listado de grupos nuevos encontrados en la red	Correcto
Obtención de un listado de contenidos que se ajustan a una búsqueda según palabra clave y grupo	Correcto
Obtención de un listado de *peers* relacionados con un grupo y una palabra clave dadas	Correcto
Búsqueda general de un contenido en las tablas	Correcto

Descripción de la Prueba	Resultado
locales y en la red por palabra clave y grupo	
Selección de un *peer* al que solicitar información	Correcto
Actualización de un valor de confianza en contenido por calificación local	Correcto
Parada del sistema	Correcto
Actualización de un valor de confianza en contenido por *feedback*	Correcto
Actualización del factor de riesgo asociado a un *peer* en la tabla local de riesgos	Correcto
Obtención del listado de grupos disponibles para la búsqueda	Correcto
Inserción de la información de contenidos en el anuncio de *peer*	Correcto
Obtención de un valor de confianza en *peer* concreto	Correcto
Obtención de un valor de confianza en contenido concreto	Correcto
Comprobación de la existencia de un par grupo-palabra clave dentro de un anuncio	Correcto
Obtención del listado de grupos contenidos en un anuncio de red	Correcto
Obtención de los grupos y palabras claves locales para añadir al anuncio de *peer*	Correcto
Obtención de un puerto libre para el funcionamiento del sistema	Correcto
Comprobación de puertos ocupados	Correcto
Almacenaje de tablas de confianza en disco	Correcto
Carga de los valores contenidos en el fichero de propiedades para la configuración del sistema	Correcto
Modificación del valor de una de las propiedades del fichero de configuración	Correcto

Descripción de la Prueba	Resultado
Obtención de los valores de las propiedades del fichero de configuración	Correcto
Obtención del contenido del fichero de *log*	Correcto
Eliminación de ficheros de *log* obsoletos	Correcto
Obtención del nombre de un contenido dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un contenido dado su nombre	Correcto
Obtención del nombre de un peer dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un peer dado su nombre	Correcto
Obtención del nombre de un grupo dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un grupo dado su nombre	Correcto
Obtención del nombre de un contenido dado su identificador *JXTA* único	Correcto
Obtención del identificador *JXTA* único de un contenido dado su nombre	Correcto
Guardado automático de tablas en disco	Correcto

Tabla 12: Pruebas del sistema completo

5.7. Pruebas de la aplicación de prueba

A pesar de haber comprobado que el funcionamiento del sistema es correcto, se debe realizar también la revisión correspondiente para verificar que la aplicación de prueba, que servirá de interfaz para que un usuario pueda utilizarlo, esté implementada correctamente. Las pruebas a las que se ha sometido son las siguientes:

Descripción de la Prueba	Resultado
Inicialización del sistema	Correcto
Navegación por los menús	Correcto
Resistencia frente a entradas no válidas	Correcto

Descripción de la Prueba	Resultado
Búsqueda de contenidos estándar	Correcto
Búsqueda de contenidos forzando consulta remota	Correcto
Creación de un nuevo contenido	Correcto
Etiquetado de un contenido con una nueva palabra clave	Correcto
Eliminación de un valor de confianza en contenido determinado	Correcto
Consulta de un valor de confianza en contenido determinado	Correcto
Actualización de la tabla de confianza en *peer*	Correcto
Consulta de un valor de confianza en *peer* determinado	Correcto
Creación de un nuevo grupo de intereses	Correcto
Búsqueda de nuevos grupos de intereses	Correcto
Consulta de *log*	Correcto
Borrado de ficheros de *log* obsoletos	Correcto
Modificación del valor de una propiedad en el fichero de configuración	Correcto
Consulta de los valores de las propiedades del fichero de configuración	Correcto

Tabla 13: Pruebas de la aplicación de prueba

5.8. Pruebas de robustez

En los apartados anteriores se ha tratado de verificar que el sistema, a lo largo de su normal funcionamiento, se comporta correctamente y se obtienen de él los resultados esperados, algo para lo que se ha trabajado a lo largo de todo el desarrollo para cumplir con los objetivos del proyecto.

Sin embargo, hay ocasiones en las que el sistema puede enfrentarse a situaciones inesperadas, que deberá solucionar y seguir funcionando sin fallo alguno. Por este motivo se ha añadido al banco de pruebas inicial, un conjunto de verificaciones que están dirigidas a comprobar cualquier comportamiento anómalo del sistema en estas situaciones. Son las siguientes:

Descripción de la Prueba	Acción	Resultado
Inexistencia del directorio personal de usuario	Se crea un nuevo directorio	Correcto
Inexistencia del fichero de tablas de confianza en contenido	Se crea un fichero nuevo	Correcto
Formato del fichero de tablas de confianza en contenido no válido	Se omite el fichero y se genera uno nuevo	Correcto
Inexistencia del fichero de tablas de confianza en *peer* dependiente de grupo	Se crea un fichero nuevo	Correcto
Formato no válido del fichero de tablas de confianza en *peer* dependiente de grupo	Se omite el fichero y se genera uno nuevo	Correcto
Inexistencia del fichero de tablas de confianza en *peer* independiente de grupo	Se crea un fichero nuevo	Correcto
Formato no válido del fichero de tablas de confianza en *peer* independiente de grupo	Se omite el fichero y se genera uno nuevo	Correcto
Inexistencia del fichero de tablas de riesgo	Se crea un fichero nuevo	Correcto
Formato no valido del fichero de tablas de riesgo	Se omite el fichero y se genera uno nuevo	Corecto
Elección de opciones no válidas en la aplicación de prueba	Se solicita de nuevo la elección	Correcto
Inexistencia del fichero de configuración	Valores por defecto	Correcto
Inexistencia de alguno de los parámetros en el fichero de configuración	Valores por defecto para los que falten	Correcto
Formato no válido de algún parámetro del fichero de configuración	Valores por defecto para los erróneos	Correcto
Introducción de caracteres no válidos en la aplicación de prueba	Se solicita de nuevo la entrada	Correcto

Tabla 14: Pruebas de robustez

CAPÍTULO 6: HISTORIA DEL PROYECTO

6.1. Introducción

En capítulos anteriores, como los correspondientes a la descripción y la implementación del sistema, se pudo ver el resultado final de todo el trabajo llevado a cabo para la consecución del mismo; esto podría llevar a la conclusión de que la implementación ha sido más sencilla de lo que parece.

Sin embargo, este resultado no se alcanzó a partir del planteamiento original del sistema, sino que a lo largo de su realización surgieron multitud de problemas a los que hubo que dar solución. Del mismo modo, también hubo que recurrir a replanteamientos de la idea original, motivados por errores en la implementación o simplemente para conseguir optimizar el sistema.

A lo largo de este capítulo se irán detallando todos los cambios relevantes a los que se sometió el sistema durante su implementación, detallando el coste temporal que esto supuso y mostrando las soluciones adoptadas finalmente. La estructura de las siguientes secciones es la siguiente:

- Evolución del sistema.
- Evolución en las estructuras básicas.
- Evolución de la comunicación.

Antes de comenzar con las explicaciones, cabe decir que se ha utilizado en las siguientes secciones un modelo de seguimiento basado en la evolución de la funcionalidad del sistema, más que en la evolución temporal del mismo. Esto es debido a que, a consecuencia de los cambios experimentados por la plataforma *JXTA*, se han tenido que modificar varios puntos de la funcionalidad, lo que ha hecho muy complicado computar el tiempo exacto dedicado a cada tarea.

6.2. Evolución del sistema

6.2.1. Plataforma JXTA

En primer lugar se debe hacer una presentación del marco en el que se desarrolló la implementación, es decir, la evolución de las tecnologías que utilizaron, dando una idea de cómo afectaron al planteamiento inicial.

Al comienzo de este proyecto, la primera decisión que se tomó fue la utilización de la plataforma *JXTA* como eje vertebrador del sistema, por lo que será una buena idea presentar los cambios que ha sufrido ésta a lo largo del tiempo, cómo se ha intentado adaptar el sistema a la evolución de la misma y los resultados obtenidos en tal empeño.

Cuando se planteó este proyecto, la última versión estable de *JXTA* era la 2.5, una implementación bastante robusta sobre la que se realizaron los primeros estudios y se hicieron las pruebas originales de conectividad entre nodos en la red. Los resultados fueron satisfactorios, permitiendo una total interconexión entre los *peers*, lo que confirmo la decisión acertada de la plataforma.

Pronto apareció una nueva versión de la plataforma, la 2.6, en la que se pudieron observar numerosos cambios que facilitarían la implementación de este sistema de gestión de la reputación. En primer lugar se dio un salto importante en los mecanismos de comunicación entre nodos, perfeccionando los protocolos de intercambio de mensajes. Esto supuso una simplificación en el paso de mensajes entre los *peers*, por lo que resultó una actualización acertada. Como aliciente extra, se vio que el paso a la nueva versión de la plataforma *JXTA* no suponía un gran cambio en otros aspectos del planteamiento inicial del sistema.

Por último, en las últimas semanas de ejecución del proyecto se publicó una nueva versión de esta plataforma, la 2.7RC1, que en principio no suponía grandes cambios con respecto a la anterior, salvo por el hecho de presentar un nuevo sistema de paso de mensajes utilizando *TLS*. La incorporación de este sistema de mensajería segura entre nodos suponía un gran avance en el sistema, por lo que se intentó adaptar la implementación llevada a cabo a esta nueva versión. Pronto se pudo comprobar que la nueva entrega de *JXTA* no era una versión estable, sino que se trataba de una versión de pruebas para que los desarrolladores pudiesen encontrar errores, por lo que, como cabía esperar, los intentos por actualizar la versión de *JXTA* no fueron fructíferos. Esto supuso una vuelta atrás a la versión previa.

A fecha de la finalización de este proyecto no se había publicado una versión estable de la entrega 2.7, por lo que considera que este sistema de gestión de la reputación utiliza la última versión plenamente funcional de *JXTA*.

6.2.2. Sistema de gestión

Una vez elegida la plataforma sobre la que descansará el sistema de gestión de la reputación, se deben establecer un conjunto de pautas básicas que definan dicho sistema. En la fase de diseño del sistema y búsqueda de posibles soluciones se barajó la posibilidad de construir un sistema de gestión de la reputación desde cero, pero la idea se rechazó, dado el elevado número de formulaciones teóricas existente al respecto.

Entre estas posibilidades nos llamó la atención la especificación del sistema *Poblano*, propuesta que escaló posiciones como candidata debido a su definición, que contaba con algoritmos suficientemente sencillos para evitar que el sistema de gestión de la reputación tuviese demasiada complejidad. La razón definitiva para elegir este sistema fue su predisposición para integrarse con la plataforma *JXTA*.

Sin embargo, a pesar de tener como base el sistema de gestión de la reputación presentado por *Poblano*, nuestro sistema no se ciñe a estas especificaciones, sino que trata de utilizar los aspectos más interesantes de éste sumados a otra parte de implementación propia, tal y como se irá viendo en las sección siguientes.

6.3. Evolución de las estructuras básicas

6.3.1. Reputación de contenidos

Si se consulta la especificación original de *Poblano*, se puede comprobar que las estructuras básicas para la gestión de la reputación son exactamente las mismas que las empleadas en este proyecto, ya que proporcionan los mecanismos suficientes para que éste sea completamente funcional. Sin embargo, dicha especificación se limita a describir los componentes de la reputación de una forma demasiado superflua, haciendo necesaria la introducción de nuevos elementos que los definan mejor.

Por ejemplo, en el caso de los valores de reputación de contenido, la especificación de *Poblano* únicamente hace referencia al identificador propio del contenido, la palabra clave a la que hace referencia, valores de popularidad y reputación, e indicador de contenido local. Como se ha mencionado anteriormente, estos parámetros serían suficientes para la utilización de la reputación de contenidos dentro del sistema de gestión.

Sin embargo, para la implementación real de un sistema de gestión de la reputación se deben añadir dos elementos más: el primero de ellos sería el identificador del grupo en el que tiene sentido dicho valor de reputación y el segundo una marca de tiempo que indicará cuándo fue creado o modificado dicho valor. Estos nuevos elementos se tuvieron en cuenta desde el primer momento del diseño del sistema, por lo que no supuso ningún tiempo extra en su desarrollo.

Por el contrario, en la recta final de la implementación se añadieron dos campos más a estos valores de reputación. El primero de ellos es el correspondiente al nombre del contenido, que servirá para que la representación de los contenidos sea más amigable, ya sea para una aplicación externa o para un usuario de la aplicación de prueba. Con el mismo objetivo se añadió el elemento correspondiente al nombre del grupo al que hace referencia dicho valor de reputación. Estos dos campos supusieron un cambio para el sistema en tres niveles: el código de la aplicación, el formato de los ficheros de almacenamiento de las tablas de confianza en contenidos y la estructura de los mensajes intercambiados entre los nodos.

En el ámbito del código hubo que añadir los atributos correspondientes a los nuevos elementos, así como los métodos que se encargan de su gestión. Para hacer este cambio efectivo hubo que modificar tanto la clase que modela el valor de reputación en contenido, como la correspondiente a la tabla de confianza en contenidos, sin olvidar por supuesto el gestor de tablas. La evolución de esta estructura del sistema de gestión de la reputación puede verse en la figura 58.

Como era de esperar, el fichero de almacenamiento de esta tabla de confianza en contenidos también tuvo que ser modificado para almacenar los nuevos atributos de la clase, hecho que se consiguió con la simple adición de nuevos elementos *XML* en el fichero correspondiente.

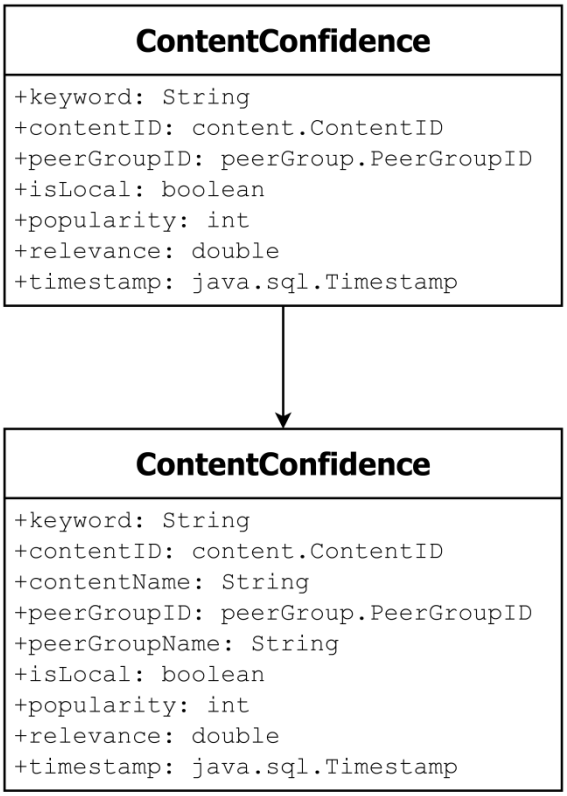


Figura 58: Evolución de la reputación de contenido

De las modificaciones llevadas a cabo en los mensajes intercambiados en el sistema se hablará más adelante, en la sección que describe la evolución de las comunicaciones.

Un último apunte que se debe hacer en cuanto a la reputación de contenidos tiene que ver con la versión de la plataforma *JXTA* utilizada en cada momento del desarrollo, y afecta directamente a la forma en la que se representan los identificadores únicos de los contenidos. En la versión 2.5 de la plataforma se utilizó para este fin una clase llamada `CodatID`, que en las versiones posteriores desapareció, sustituyéndose por otra denominada `ContentID`, con mayor funcionalidad y capacidad de representación de identificadores. Por este motivo, se hizo necesaria la migración a esta nueva clase suponiendo un retraso de unos tres días en la ejecución del proyecto.

6.3.2. Reputación de peers

Al igual que se comentaba en el apartado anterior, los valores de reputación en *peers* también han sufrido modificaciones a lo largo de la implementación del sistema de gestión de la reputación.

Ya desde el comienzo del desarrollo, hubo que añadir a los elementos esenciales descritos en la especificación de *Poblano* un nuevo elemento: la marca de tiempo que indica el momento en el que dicho valor de reputación fue creado o modificado por última vez. Como se puede ver, esto no supuso tanta diferencia como en el caso de los valores de reputación de contenido, en los que hubo que añadir más elementos.

Pronto se vio la necesidad de añadir otros dos nuevos atributos: el primero de ellos se corresponde con el número de contenidos que se han recibido del *peer* al que hace referencia el valor de reputación, y el segundo se trata de la suma de todos estos valores de reputación. Estos dos nuevos elementos de la reputación de *peer* son esenciales para el cálculo de los umbrales de cooperación de cada uno de los *peers* con los que se desea intercambiar información.

La introducción de estos dos nuevos elementos supuso la modificación del formato de las tablas de confianza en *peer* y, por lo tanto, también del fichero *XML* correspondiente, lo que supuso un retraso de un par de días en el proyecto.

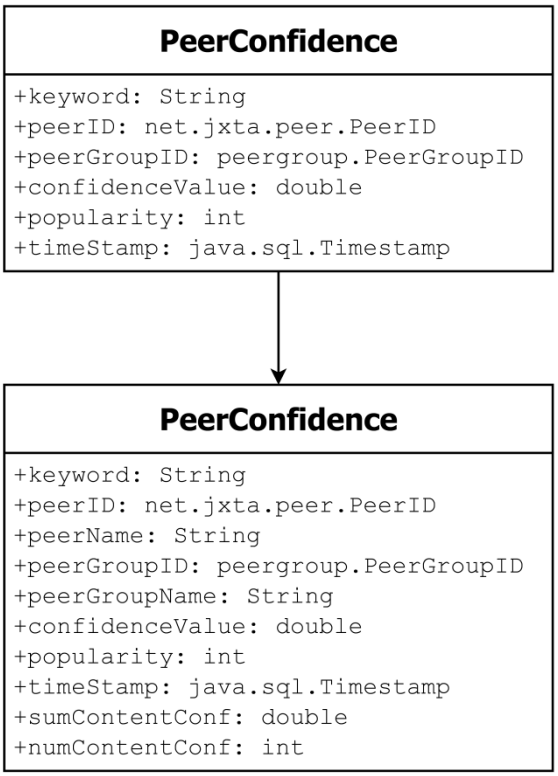


Figura 59: Evolución de la reputación de peer

Tal y como ocurrió con la reputación de contenido, en la recta final de la implementación del proyecto se encontró la necesidad de incluir en esta estructura dos nuevos elementos, que facilitarían la representación de la información de cara al usuario. En primer lugar se tuvo que añadir una cadena de caracteres que complementase al identificador único del *peer* al que hace referencia la reputación; esta cadena de caracteres representará el nombre del *peer*. Por otro lado se añadió otra cadena de caracteres que indica el nombre del grupo al que hace referencia dicho valor.

La evolución completa de la reputación de *peer* se puede ver en la figura 59 y, como era de esperar, supuso cambios en el proyecto tanto a nivel de código, ya que la clase correspondiente a estos valores de reputación, así como las que modelan las tablas que agrupan estos valores, tuvieron que ser modificadas para añadir dichos atributos y los métodos que los manejan.

Por otro lado estos cambios también supusieron el formato del fichero *XML* que alberga las tablas en disco, por lo que se tuvo que modificar el mecanismo de lectura y guardado del mismo. Cuando se vio la necesidad de estos cambios ya se estaba sobre aviso por las modificaciones similares producidas en los valores de reputación en contenido, por lo que no supuso un retraso de más de un día hacer las correcciones necesarias.

6.3.3. Factor de riesgo

Esta es la clase que menos modificaciones ha sufrido a lo largo de la implementación del proyecto. En un comienzo, los elementos constituyentes del factor de riesgo especificados por *Poblano* fueron suficientes para acometer el trabajo, y esta fue la única de las estructuras que se mantuvo inalterada.

Sin embargo, y tal y como ocurrió con las dos estructuras anteriores, se vio la necesidad de completar el identificador del *peer* al que hace referencia el factor de riesgo con una cadena de caracteres que representase su nombre. Todo esto se produjo por la necesidad de presentar toda la información de forma más amigable. La evolución de esta estructura puede verse en la figura 60.

En este caso nos encontramos con el mismo problema que antes, es decir, el cambio no solo de la clase que modela esta estructura, sino también las correspondientes a la tabla de riesgos, así como el fichero de almacenado de esta tabla en disco. A pesar de esto, los cambios no supusieron un retraso en el proyecto de más de una jornada de trabajo.

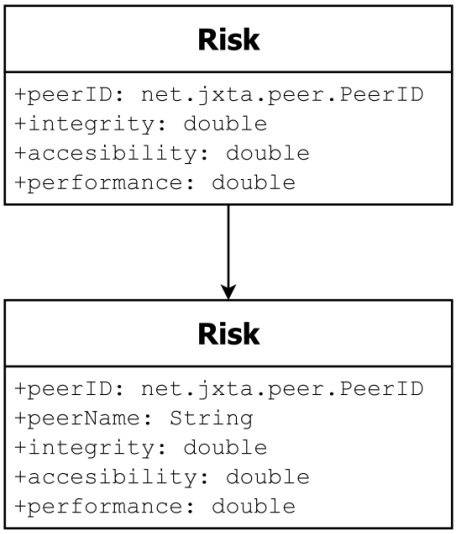


Figura 60: Evolución del factor de riesgo

6.4. Evolución de la comunicación

La comunicación entre los *peers* ha sido uno de los puntos que ha generado más problemas a lo largo del desarrollo del proyecto. Esto, en principio, se debió al desconocimiento de la plataforma *JXTA*, y por tanto de los mecanismos que proporcionaba para la comunicación entre los nodos de la red.

Una vez estudiadas las posibilidades ofrecidas por la plataforma, el problema fue elegir la opción más apropiada para implementar las comunicaciones que se llevarían a cabo entre los nodos, es decir, el mecanismo idóneo para la transmisión de los mensajes que debe soportar el sistema.

En primer lugar se barajó la posibilidad de que la aplicación lanzase un hilo cuyo cometido fuese, exclusivamente, permanecer a la escucha de nuevos mensajes entrantes para su procesado. Esta idea se descartó, debido a la complejidad y la falta de eficiencia de la solución. Por este motivo, se siguió haciendo pruebas hasta dar con la que finalmente se adoptó.

La solución final pasó por la utilización de un mecanismo propio de la plataforma *JXTA*, concretamente el servicio de descubrimiento de peers (*Discovery Service*) que proporciona una gestión de mensajes entrantes transparente para el desarrollador, que no tiene que preocuparse de la espera de nuevas conexiones.

Una vez definido el mecanismo de comunicación básico, lo único de lo que había que preocuparse era de que tanto los anuncios de red, como los mensajes intercambiados, incluyeran la información necesaria para el correcto funcionamiento del sistema, temas que se tratarán en los dos siguientes apartados.

6.4.1. Mensajes

Las sucesivas actualizaciones de la plataforma *JXTA* supusieron cambios importantes a la hora de establecer la comunicación entre nodos de la red. Como primera opción, durante la utilización de la versión 2.5 de *JXTA*, se optó por la utilización de tuberías bidireccionales para este cometido.

Las tuberías bidireccionales suponían un mecanismo de comunicación de bajo nivel en el que se hacía necesaria la gestión directa de múltiples factores para el establecimiento de la conexión. Entre estos factores se podían enumerar desde la espera a que el canal de comunicación estuviese establecido, hasta la conversión de los datos para enviar, lo que suponía un nivel de complejidad en el código bastante alto.

Cuando se lanzó la versión 2.6 de la plataforma *JXTA* se introdujeron numerosos cambios y mejoras en cuanto a la comunicación entre nodos. En concreto, se amplió el servicio específico de enrutamiento entre extremos (*Enpoint Service*) con un sistema gestor de mensajes denominado *Messenger*. Este mecanismo de comunicación de alto nivel evitaba tener que preocuparse por los aspectos básicos de la comunicación necesarios en las tuberías, gestionándolos directamente.

Gracias al *Messenger* lo único de lo que había que preocuparse, era de contar con el identificador único del *peer* con el que se quería entablar la comunicación para establecer el canal por el que se enviarían los mensajes. Los mensajes, a su vez, pasaron a convertirse en simples estructuras *XML* mucho más fáciles de construir y manejar, lo que supuso una simplificación de ciertas secciones del código.

Por último, tal y como se ha mencionado en la sección de evolución del sistema, se barajó la posibilidad de migrar a la versión 2.7 de la plataforma *JXTA*, dada la introducción de un sistema gestor de mensajes, es decir, un *Messenger*, basado en las comunicaciones seguras definidas por *TLS*. Como se ha dicho esto fue inviable dado que la versión 2.7RC1 lanzada no era una versión estable.

El conjunto de esta evolución en cuanto al sistema de comunicación entre nodos a través de mensajes, originó un retraso de aproximadamente una semana en la implementación del sistema de gestión de la reputación, ya que supuso la modificación de las clases que modelan los mensajes así como la estructura de la clase principal.

En este tiempo de retraso también se debe incluir el dedicado a la modificación de la estructura de los mensajes intercambiados entre los nodos de la red, ocasionado por la evolución de las estructuras básicas. La introducción de nuevos elementos, como las cadenas de caracteres para representar los nombres de contenidos, *peers* y grupos, descubrió la necesidad de incluirlos en estos mensajes.

De esta forma, los mensajes, que en principio eran estructuras moderadamente simples y con un tamaño relativamente pequeño, se vieron obligadas a almacenar un volumen de datos mayor y con una estructura ligeramente diferente.

Antes de que los mensajes llegasen a tener la forma propuesta en el capítulo de descripción del sistema, es decir, en el planteamiento inicial del sistema, ninguno de ellos necesitaba incluir campos como los correspondientes a los nombres de contenidos, *peers*, o grupos. La evolución más significativa se puede encontrar en el mensaje de respuesta a una búsqueda de contenidos general, aunque es extensiva al resto de tipos de mensaje. En la figura 61 se muestran los cambios sufridos por esta estructura.

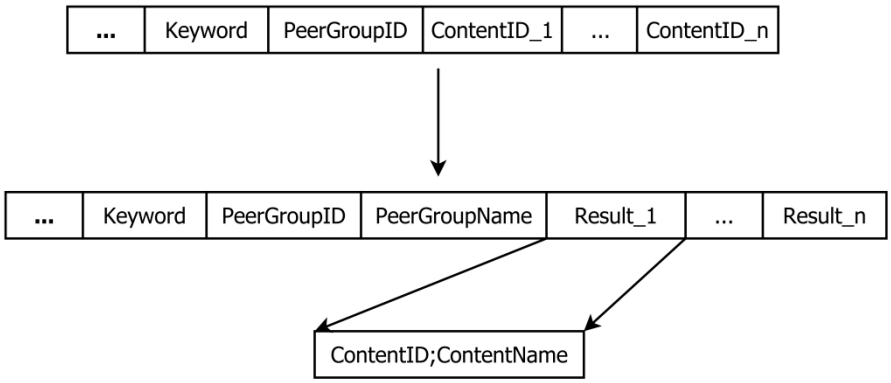


Figura 61: Evolución de un mensaje de búsqueda

En esta figura puede verse cómo afectó la aparición de los nuevos atributos a un tipo concreto de mensaje, pero debe tenerse en cuenta que también tuvieron que ser modificados, en mayor o menor medida, el resto de tipos de mensaje, así como la parte común a todos ellos. Este cambio, aunque importante en cuanto a la variación de los datos intercambiados entre los nodos, no afectó al rendimiento general del sistema y supuso un pequeño incremento en el volumen de datos intercambiado entre nodos que puede ser despreciable.

6.4.2. Anuncios

Cuando se inició el estudio sobre la tecnología *JXTA* se pudo comprobar el alto grado de interconexión con *XML*, estando basado en esta especificación. Esto hacía necesario el conocimiento de todas las estructuras presentes en la plataforma, como podían ser los anuncios que se publicaban en la red cada vez que un nodo accedía al sistema.

Aunque en un primer momento los anuncios se limitaron a simples mecanismos de detección de otros nodos de la red, pronto se vio su potencial para propagar información relativa a nuestro sistema de gestión de la reputación. Al comienzo, el sistema de búsqueda de contenidos en la red se basaba simplemente en los pasos definidos en la especificación de *Poblano*, procedimiento robusto pero que aún podía ser mejorado.

El carácter modificable y configurable de los anuncios de *peer* en la plataforma *JXTA* supuso una evolución en el mecanismo de búsqueda del sistema de gestión de la reputación, tal y como se pudo comprobar en el capítulo dedicado a la descripción general del sistema. La especificación original se ceñía a búsquedas en las tablas locales y peticiones a los *peers* almacenados en las mismas, obviando la existencia de más nodos dentro de la red, lo que implicaba una búsqueda incompleta. Por este motivo se propuso un tercer paso en la búsqueda: la consulta de los anuncios de *peer* obtenidos de la red.

Este nuevo paso en el sistema de búsqueda no conllevó una gran diferencia en cuanto al procedimiento con el que las búsquedas se hacían, ya que la introducción de la búsqueda en anuncios supuso un simple anexo al mecanismo original. El cambio más importante en el sistema de gestión de la reputación, fue la necesidad de introducir nuevos métodos para el procesado de dichos anuncios y la obtención de la información contenida en ellos.

De nuevo aquí jugó un papel importante la estructura *XML* de dichos anuncios, ya que esta naturaleza facilitó mucho la implementación de los mecanismos para su procesamiento. Por este motivo, el cambio no supuso un retraso en la implementación de más de cuatro días.

CAPÍTULO 7: CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Conclusiones

Una vez terminado todo el trabajo de diseño e implementación del sistema sobre el que trata este proyecto, se ha podido hacer balance de los objetivos marcados que finalmente han sido cumplidos. Del mismo modo, se pueden enumerar una serie de conclusiones obtenidas de todo este proceso. Serán las siguientes:

- Se ha descubierto la tecnología *JXTA* como una plataforma lo suficientemente robusta como para poder soportar el desarrollo de cualquier tipo de aplicación *Peer-to-Peer*, sean cuales sean las necesidades iniciales para su implementación.
- Se ha podido comprobar que las directrices establecidas por la especificación de *Poblano* para la implementación de un sistema de gestión de la reputación han sido muy útiles para el desarrollo de este proyecto, pero aún así se ha visto que se pueden perfeccionar para obtener un sistema con un diseño más funcional.
- Se ha confirmado con creces el potencial de una tecnología suficientemente desplegada, como es *XML*, cuya utilidad es indiscutible, y que está por encima de las aplicaciones sobre las que se utilice, formando la base de la plataforma *JXTA* y siendo pieza clave del sistema de gestión de la reputación implementado en este proyecto.
- Se ha comprobado la viabilidad de unificar las tres tecnologías, *JXTA*, *XML*, y las redes *Peer-to-Peer*, con la especificación de *Poblano*, para la implementación de un sistema de gestión de la reputación completamente descentralizado y plenamente funcional.

Siguiendo en la misma línea, se debe hacer balance de los logros alcanzados en la implementación en función de los requisitos fundamentales que se especificaron en la descripción del sistema. Se pueden resumir en los siguientes:

- Se ha podido hacer un amplio estudio de las posibilidades de comunicación entre los nodos de una red *P2P*, evaluando las distintas alternativas y aprovechando las ofrecidas por las tecnologías utilizadas.
- Se ha implementado un sistema de comunicación entre nodos basado en un protocolo de petición y respuesta, en el que se ha debido definir por completo el formato de los mensajes. Esto ha permitido acercarse a la problemática del diseño de un protocolo desde cero.
- Se ha dotado al sistema de un mecanismo de almacenamiento automático de toda la información de reputación para evitar al usuario preocupaciones innecesarias por pérdida de datos.

- Se ha implementado en el sistema un mecanismo de decisión que le permitirá seleccionar a un nodo de la red como fuente de información en base a la reputación que posea, evitando a su vez la sobrecarga de los nodos que mejor comportamiento tengan.
- Se ha implementado un sistema de cálculo de umbrales de cooperación, que permitirá al nodo evaluar el grado de predisposición que tiene un peer de colaborar en base a su reputación y su rendimiento, evitando así la acción de posibles nodos maliciosos.
- Se ha incluido la posibilidad de configuración del sistema mediante un fichero de propiedades que permitirá una gran flexibilidad en cuanto al comportamiento del nodo, de forma que se pueda ajustar el rendimiento del mismo.
- Se ha dotado al sistema de una aplicación de administración que permitirá a un usuario comprobar la funcionalidad del sistema, así como realizar tareas administrativas como la configuración de los parámetros que determinarán el comportamiento del *peer*.
- Se ha incluido en el sistema un módulo de gestión de *logs* que permitirá realizar tareas de supervisión para verificar el correcto funcionamiento del sistema de una forma inteligible para el usuario.
- Se ha dotado al sistema de una interfaz que permitirá a cualquier aplicación externa el acceso a toda la funcionalidad del sistema de gestión de la reputación, ampliando así la funcionalidad del proyecto.

7.2. Líneas futuras de trabajo

Aunque los objetivos marcados para el desarrollo de este proyecto han quedado satisfechos en mayor o menor medida, y se ha conseguido hacer funcionar un sistema de gestión de la reputación suficientemente robusto y funcional, debemos ser conscientes de que ha habido aspectos del mismo que pueden ser mejorables, así como pueden existir nuevas funcionalidades adicionales al planteamiento inicial.

Todas estas características pueden ser resumidas en líneas generales que podrán ser tenidas en cuenta en futuros desarrollos y ampliaciones del sistema, y que han resultado ser interesantes. Se resumen en cinco líneas básicas:

- La integración del sistema de gestión de la reputación con un sistema de almacenaje de las tablas de confianza basado en bases de datos. Este planteamiento será válido únicamente para aquellos dispositivos en los que los recursos no estén limitados, siendo transparente para las comunicaciones entre *peers*.
- El desarrollo de un sistema de comunicación entre nodos basado en la tecnología *TLS*, aportando de este modo seguridad en el intercambio de información entre nodos. Como se ha visto, futuras versiones de la

plataforma *JXTA* contarán con este mecanismo, por lo que será una línea de desarrollo sencilla.

- Implementación del sistema de gestión de la reputación tomando como punto de partida la implementación *JXTA-C* con el objetivo de comprobar el rendimiento del sistema de gestión de la reputación basado en el lenguaje de programación *C*.
- Estudio del rendimiento del sistema de gestión de la reputación en dispositivos con recursos limitados, como pueden ser teléfonos móviles o *PDA*, a través de la implementación del sistema de gestión de la reputación con la plataforma de *JXTA* móvil, *JXME*.
- Pruebas exhaustivas con un sistema de anotación semántica desplegado para comunidades.

Bibliografía

[1] Universidad Carlos III de Madrid; Universidad Politécnica de Cataluña, «ITACA: Anotación Semántica y Formación de Comunidades de Usuarios Basadas en Consultas y Soportadas por un Modelo de Confianza,» [En línea]. Available: http://webtlab.it.uc3m.es/projects/ITACA. [Último acceso: 15 Mayo 2012].

[2] R. Schollmeier, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications.,* 2001.

[3] S. Androutsellis-Theotokis y D. Spinellis, *A Survey of Content Distribution Technologies,* ACM Computing Surveys, 2004.

[4] I. Clarke, O. Sandberg, B. Wiley y T. Hong, *Freenet: A Distributed Anonymous Information Storage and Retrieval System. Freenet White Paper.,* 1999.

[5] Varios, «I2P Anonymous Network,» [En línea]. Available: http://www.i2p2.de/. [Último acceso: 12 Febrero 2012].

[6] J. Buford, H. Yu y E. Lua, *P2P Networking and Applications,* Morgan Kauffman, 2008.

[7] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek y H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications,* 2001.

[8] Massachusetts Institute of Technology, «Massachusetts Institute of Technology,» [En línea]. Available: http://web.mit.edu/. [Último acceso: 26 Mayo 2012].

[9] S. Ratsanamy, P. Francis, M. Handley, R. Karp y S. Shenker, *A Scalable Content-Addressable Network,* 2001.

[10] P. Maymounkov y D. Mazières, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric,* 2002.

[11] M. Steiner, T. En-Najjary y E. W. Biersack, *A Global View of Kad,* 2007.

[12] R. Stern, «Napster: A Walking Copyright infringement?,» *IEEE micro,* 11 2000.

[13] Kazaa.com, «Kazaa,» [En línea]. Available: www.kazaa.com. [Último acceso: 12 Febrero 2012].

[14] Varios, « BitTorrent Especification,» [En línea]. Available: http://www.bittorrent.org. [Último acceso: 14 Febrero 2012].

[15] uTorrent, «uTorrent: A (very) tiny BitTorrent Client,» [En línea]. Available: http://www.utorrent.com. [Último acceso: 1 Junio 2012].

[16] Varios, «General Gnutella / Gnutella Network discussion,» [En línea]. Available: http://www.gnutellaforums.com. [Último acceso: 14 Febrero 2012].

[17] «Gnutella2 Developer Network,» [En línea]. Available: http://g2.trillinux.org. [Último acceso: 9 Marzo 2012].

[18] Sun Microsystems, JXTA Java Standard Edition v2.5: Programmers Guide, 2007.

[19] Varios, «The Internet Engineering Task Force (IETF),» [En línea]. Available: http://www.ietf.org/. [Último acceso: 12 Mayo 2012].

[20] IRTF, «Internet Research Task Force,» [En línea]. Available: http://www.irtf.org/p2prg. [Último acceso: 12 Marzo 2012].

[21] Varios, «JXTA: The Language and Platform Independiente Protocol for P2P Networking,» [En línea]. Available: http://jxta.kenai.com. [Último acceso: 25 Mayo 2012].

[22] Sun Microsystems, JXTA Protocol Specification, 2007.

[23] S. Kamvar, M. Schlosser y H. García-Molina, *The EigenTrust Algorithm for Reputation Management in P2P Networks,* 2003.

[24] A. Singh y L. Liu, *TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems,* 2003.

[25] P. Michiardi y R. Molva, *CORE: A Collaborative Reputation Mechanism to enforce node cooperation in Mobile Ad-hoc Networks,* 2001.

[26] R. Chen y W. Yeager, *Poblano: A distributed Trust Model for Peer-to-Peer Networks,* 2001.

[27] eBay, «eBay,» [En línea]. Available: http://www.ebay.com. [Último acceso: 15 Mayo 2012].

[28] Amazon, «Amazon,» [En línea]. Available: http://www.amazon.es. [Último acceso: 15 Mayo 2012].

[29] Google, «Google,» [En línea]. Available: http://www.google.com. [Último acceso: 15 Junio 2012].

[30] B. Snifen, *Trust Economies in the Free Haven Project,* 2000.

[31] M. Waldman, A. Rubin y L. Cranor, *Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System,* 2000.

[32] World Wide Web Consortium, [En línea]. Available: http://www.w3.org/. [Último acceso: 29 Mayo 2012].

[33] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 Recommendation (Fifth Edition),* 2008.

[34] International Standard Organization, *ISO 8879: Information Processing - Text and office systems - Standard Generalized Markup Language (SGML) Correction 2,* 1999.

[35] International Standard Organization, *ISO/IEC 15445: Information Technology - Document description and processing languages - Hiper Text Markup Language (HTML),* 2000.

[36] World Wide Web Consortium, *Extensible Markup Language (XML) 1.1 Recomendation (Second Edition),* 2006.

[37] International Standard Organization, *ISO/IEC 14977: Information Technology - Syntactic metalanguage - Extended BNF,* 1996.

[38] World Wide Web Consortium, «XSL Transformations (XSLT) Version 2.0 (Second Edition),» 21 4 2009. [En línea]. Available: http://www.w3.org/TR/xslt20/. [Último acceso: 12 02 2012].

[39] World Wide Web Consortium, «W3C XML Schema Definition Language (XSD) 1.1: Structures,» 19 01 2012. [En línea]. Available: http://www.w3.org/TR/xmlschema11-1/. [Último acceso: 12 02 2012].

[40] World Wide Web Consortium, «XML Linking Language (XLink) Version 1.1,» 06 05 2010. [En línea]. Available: http://www.w3.org/TR/xlink11/. [Último acceso: 12 02 2012].

GLOSARIO

A

Anuncio (*Advertisement*): Estructura de datos *XML* utilizada por la plataforma *JXTA* para la publicación de recursos en la red.

API (*Application Programming Interface*): Librería de funciones que proporciona a los programadores mecanismos para realizar tareas comunes como la transferencia de ficheros, la conectividad de red y la implementación de estructuras de datos.

ACK (*Acknowledgement*): Aviso enviado entre dos nodos de la red para confirmar la recepción correcta de una determinada información.

B

***Bootstrapping*:** Mecanismo utilizado en entornos informáticos y de programación mediante el cual un sistema muy sencillo es capaz de activar o arrancar otros sistemas más complejos.

Booleano: En la mayoría de los lenguajes de programación, tipo de datos que puede tomar valores "verdadero" o "falso", ideado para la representación de datos lógicos.

C

Confianza: Relación entre dos nodos de la red que determina el grado de predisposición que uno de ellos presenta para ayudar al otro. Si el término se aplica a un determinado contenido, la confianza indicará la valoración que tiene un determinado nodo del mismo.

Contenido: Cualquier tipo de recurso susceptible de ser compartido en la red, y que estará identificado unívocamente dentro de esta.

***ContentConfidence*:** Valor de confianza que un nodo deposita en uno de los contenidos compartidos en la red. Los valores de confianza en contenidos se agruparán en tablas para facilitar su manejo.

D

DHT (*Distributed Hash Table*): Diseño de sistema distribuido que proporciona un sistema de búsqueda basado en pares clave-valor, y en la que cualquier nodo participante puede recuperar de forma eficiente el valor asociado con una clave dada.

DHCP (*Dynamic Host Configuration Protocol*): Protocolo basado en *BOOTP* que proporciona un marco común para la distribución de información de configuración entre equipos de una red *TCP/IP*.

DNS (*Domain Name System*): Sistema de nomenclatura jerárquica para equipos informáticos, servicios o cualquier recurso conectado a Internet o a una red privada. Su función más importante es la de traducir nombres inteligibles para los humanos en identificadores binarios asociados con los equipos conectados a la red.

DTD (*Document Type Definition*): Descripción de la estructura y sintaxis de un documento *XML* o *SGML*. Su función básica es la descripción de la estructura de datos común que mantenga la consistencia entre todos los documentos que utilicen la misma definición.

E

***Endpoint*:** Punto de terminación de red utilizado por la plataforma *JXTA* para realizar representaciones virtuales de dos extremos de una conexión.

F

***Free Rider*:** Problema producido en las redes *Peer-to-Peer* por usuarios que utilizan recursos de red ofrecidos por otros nodos pero que no comparten sus propios recursos, tal y como se espera de una red de este tipo.

***Firewall*:** Dispositivo, o conjunto de dispositivos, diseñado para permitir o denegar el paso de información por la red basándose en unas determinadas reglas. Es utilizado frecuentemente para prevenir accesos no autorizados a redes y permitir el paso de comunicaciones legítimas.

***Feedback*:** Mecanismo de realimentación en el que un nodo recibe información del resultado de una acción pasada, realizada con otro nodo, para determinar su futuro comportamiento.

Fungible, material: Término utilizado principalmente en derecho civil que hace referencia a las cosas y/o bienes que se deterioran o destruyen con el tiempo, al hacer uso de ellos.

G

Gantt, Diagrama de: Herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado, como puede ser la duración de un proyecto.

H

HTML (*HyperText Markup Language*): Es el lenguaje predominante para la publicación de páginas Web en la red. Es un formato no propietario basado en *SGML* que puede ser creado y procesado por multitud de herramientas.

I

IETF (*Internet Engineering Task Force*): Organización internacional abierta de normalización cuyo cometido es hacer que el funcionamiento de Internet sea mejor, proporcionando documentos técnicos de alta calidad que influyan en la forma que la gente diseña, utiliza, y gestiona Internet.

Inundación (*Flooding*): Mecanismo utilizado en propagación de mensajes mediante el cual un paquete recibido por un nodo, es reenviado a todos los nodos con los que tiene conexión directa exceptuando aquel del que se recibió.

IRTF (*Internet Research Task Force*): Organización hermana del *IETF* cuya misión es la de promover la investigación de la evolución de Internet, creando grupos de trabajo específicos especializados en protocolos, aplicaciones, arquitectura y tecnología de Internet.

Inventariable, material: Como regla general, serán los materiales no fungibles, es decir, aquellos de los que se pueda hacer uso sin que se consuman, y que por tanto no se deterioren con el tiempo.

J

JXTA: Conjunto de protocolos abiertos independientes de plataformas y lenguajes de programación iniciado por *Sun Microsystems* en 2001 para integrar la funcionalidad de las redes *Peer-to-Peer*.

JXSE: Implementación de la plataforma *JXTA* para el lenguaje de programación *Java*, concretamente para la versión *Java SE*.

JXTA-C: Implementación de la plataforma *JXTA* para el lenguaje de programación *C*.

JXME: Implementación de la plataforma *JXTA* para el lenguaje de programación *Java*, concretamente para la versión *Java ME*.

JVM (*Java Virtual Machine*): *Software* implementado sobre *hardware*, virtual o no, que puede ejecutar código *Java*. Es el componente de ejecución de código de la plataforma de desarrollo *Java*.

Java: Lenguaje de programación estructurado, desarrollado originalmente por *Sun Microsystems* (posteriormente absorbida por *Oracle*) que forma la base de la plataforma con el mismo nombre.

K

***Keyword*:** Palabra clave utilizada para realizar las evaluaciones y calcular los valores de confianza en contenidos y *peers* en un determinado ámbito.

L

Log: Registro de toda la actividad llevada a cabo por un servidor, aplicación o software. Suele ir presentado cronológicamente con datos adicionales detallados que se utilizan generalmente con fines estadísticos y de gestión.

M

MANET (*Mobile Ad-hoc NETwork*): Estructura de red Ad-hoc que puede cambiar de localización y reconfigurarse automáticamente. Dada su naturaleza móvil, estas redes generalmente utilizan conexiones inalámbricas para interconectarse con diferentes redes.

N

Nodo: Es cada uno de los dispositivos que actúan como iguales en una red *Peer-to-Peer*. No tiene por qué ser un dispositivo físico, sino que pueden coexistir en un mismo sistema *hardware* varios nodos como entidades *software*.

NAT (*Network Address Translation*): En redes de ordenadores, es un mecanismo empleado por los *routers* para intercambiar paquetes entre dos redes que se asignan mutuamente direcciones incompatibles. Este proceso consiste en la conversión, en tiempo real, de las direcciones utilizadas en los paquetes transportados.

Ñ

O

P

***Peer*:** Véase nodo.

P2P (*Peer-to-Peer*): Topología de red en la que todos los nodos, o *peers*, participantes actúan como iguales, sin necesidad de los papeles tradicionales de cliente y servidor, ya que cada miembro actúa como ambos.

PKI (*Public Key Infraestructure*): En criptografía, es una combinación de *hardware*, *software*, políticas y procedimientos de seguridad que permiten la ejecución con garantías de operaciones criptográficas como el cifrado, la firma digital o el no repudio de transacciones electrónicas.

***Pipe*:** También conocido como tubería, es un mecanismo de transferencia asíncrono, unidireccional y fiable, utilizado para la comunicación y la transferencia de datos.

Poblano: Estudio teórico sobre la implementación de un sistema de gestión de la confianza orientado a su utilización sobre la plataforma *JXTA*.

***PeerConfidence*:** Valor de confianza que un nodo deposita en otro *peer* de la red, expresado como la predisposición que tiene el primero a interaccionar con el segundo.

Q

R

RTT (*Roundtrip Time*): En telecomunicaciones, es el tiempo que tarda una señal en ser enviada a su destino, más el tiempo que se tarda en recibir la confirmación de que realmente ha sido recibida por parte del destinatario.

***Risk*:** También conocido como riesgo, y asociado a un determinado *peer*, es un valor numérico que indica el grado de peligro que conlleva realizar una transacción con dicho nodo. Será calculado en base a las acciones pasadas de ese *peer*.

S

SGML (*Standard Generalized Markup Language*): Tecnología estandarizada por la Organización Internacional de Estándares (*ISO*) para definir lenguajes de marcado para documentos. Es el precursor de otros lenguajes como *XML* o *HTML*.

***String*:** En programación, variable que representa una cadena de caracteres.

T

TTL (*Time To Live*): En informática, es un mecanismo que limita el tiempo de vida de un dato determinado dentro de un dispositivo o una red. Puede estar implementado como un contador, o una marca de tiempo asociada al dato o incluido en él.

TLS (*Transport Layer Security*): Al igual que su predecesor *SSL* (*Secure Sockets Layer*), es un conjunto de protocolos criptográficos que ofrecen seguridad a las comunicaciones que se establecen en Internet.

U

Umbral de cooperación: Valor numérico que indica el límite mínimo que debe tener la confianza en un nodo de la red para considerar que este se mostrará dispuesto a colaborar en un futuro intercambio de información.

UML (*Unified Modeling Language*): Es un lenguaje de modelado, de propósito general y estandarizado, que se aplica en el campo de la ingeniería del *software* orientada a objetos.

V

W

W3C (*World Wide Web Consortium*): Comunidad internacional cuyas organizaciones miembro, personal completamente dedicado, y el público en general, trabajan juntos para desarrollar estándares *Web*.

X

XML (*eXtensible Markup Language*): Es un metalenguaje de marcado extensible desarrollado por el *W3C*, ideado como una simplificación y adaptación de *SGML*.

Y

Z

ANEXO A: PRESUPUESTO

A.1. Introducción

Este capítulo pretende mostrar una visión del proyecto en términos económicos, desglosando el coste de los recursos implicados en la realización de los trabajos a lo largo del tiempo, y estimando el precio de venta final que tendría este sistema en el mercado.

El presupuesto se puede dividir en los siguientes apartados que se irán desarrollando a continuación:

- Costes directos de personal
- Material inventariable
- Material fungible
- Costes generales
- Impuesto sobre el valor añadido
- Total

A.2. Fases del proyecto

En primer lugar, para comenzar con las estimaciones del proyecto, se deberán ordenar las fases por las que se ha pasado para la conclusión del mismo, así como las horas requeridas por cada una de ellas. En lugar de mostrar aquí un detalle exhaustivo de todas las tareas individuales, que se puede consultar en el anexo correspondiente, se hará un resumen por fases más general, como se puede ver en la tabla 1.

FASE	DURACIÓN (días)	DURACIÓN (horas)
Planificación	14	112
Análisis	30	240
Desarrollo	110	880
Pruebas	65	520
Documentación	50	400
Total:	**269**	**2152**

Tabla 15: Fases del Proyecto

Para el cálculo se ha tenido en cuenta una jornada laboral de 40 horas semanales, es decir, 8 horas diarias a 5 días por semana. En este caso la duración total del proyecto en días naturales será de aproximadamente 12 meses.

A.3. Costes directos de personal

En este apartado se hará una planificación del personal necesario para la ejecución del proyecto, así como su dedicación a cada una de las tareas del mismo. Esta valoración se realiza en horas de trabajo para cada uno de los participantes. En este caso concreto, los participantes en el proyecto serán:

- Jefe de Proyecto o Tutor.
- Analista/Programador o proyectando.

Tal y como se explicó en el apartado correspondiente a las tareas, la estimación de las horas de trabajo para cada uno de los integrantes del proyecto sería la siguiente:

TRABAJADOR	HORAS DE TRABAJO
Jefe de Proyecto	238
Analista / Programador	2152
TOTAL:	**2390**

Tabla 16: Horas de Trabajo por Categoría

Como se puede observar, el número total de horas trabajadas entre ambos es superior al estimado en las tareas. Esto se debe a que existen ciertas subtareas que requieren la presencia de ambos, como por ejemplo las reuniones periódicas de seguimiento en cada una de las fases.

Según las tablas estimadas de salarios para los trabajadores se establecen las siguientes remuneraciones para los participantes en el proyecto según estas categorías:

CATEGORIA	SALARIO/HORA (€)
Profesor Titular (Jefe de Proyecto)	35.86
Diplomado (Analista/Programador)	24.65

Tabla 17: Remuneración por Categoría

Por tanto, realizando la multiplicación de las horas trabajadas de cada empleado por el salario correspondiente, se obtiene que el **coste directo por personal** asciende a **61581.48 €.**

A.4. Material inventariable

En este apartado se deberá reflejar el coste total de todo el material inventariable que deba adquirirse para la realización de los trabajos. En esta categoría se puede incluir conceptos como:

- Licencias de software y mantenimiento de software.
- Equipos y mantenimiento de equipos.

En el caso de este proyecto se pueden incluir en este apartado el equipo en el que se ha realizado todo el trabajo. Al ser equipo no adquirido específicamente para dicho proyecto no se puede imputar el coste total de los mismos, sino que hay que hacer una estimación de la amortización del mismo en el periodo de realización del trabajo según su grado de utilización. La siguiente tabla muestra el desglose de dicho coste:

EQUIPO	COSTE (€)	% USO	AMORTIZACIÓN(€)
AMD Athlon	1429	50	142.90
Equipo UC3M	1299	40	103.92
Sony VAIO	1199	10	23.98
		TOTAL:	**270.80**

Tabla 18: Costes de Amortización

Para este cálculo, se ha considerado que el tiempo de dedicación de los equipos al proyecto ha sido la duración total del mismo, 12 meses tal y como se especificó en el apartado de planificación del proyecto. En cuanto al tiempo de depreciación de dichos equipos se ha estimado en 60 meses, ya que son equipos normales.

En cuanto a licencias y mantenimiento de software, no se incluye un apartado específico dedicado a los mismos, ya que se incluyen los costes correspondientes a estos conceptos dentro del coste inicial de los equipos. Por otro lado, se ha utilizado en la medida de lo posible software libre para ahorrar costes, siempre y cuando el rendimiento de dicho software fuese aceptable para el trabajo.

Por otra parte hay que considerar los costes de mantenimiento de dichos equipos. En este caso se ha estipulado este coste como el 7% anual del coste inicial del equipo. En la siguiente tabla se detalla el cálculo del importe imputable al proyecto:

EQUIPO	COSTE (€)	MANT. ANUAL (€)	MANT. TOTAL (€)
AMD Athlon	1429	100.03	100.03
Equipo UC3M	1299	90.93	90.93
Sony VAIO	1199	83.93	93.93
		TOTAL:	**284.89**

Tabla 19: Costes de Mantenimiento

Por lo tanto, el **coste total por material inventariable**, tomado como la suma del coste de amortización más el de mantenimiento de los equipos asciende a **555.69 €.**

A.5. Material fungible

En este apartado se incluyen los gastos correspondientes a equipos y materiales de vida útil corta que sean necesarios para la realización de los trabajos. Se han incluido para este proyecto en concreto los materiales de oficina (tinta de impresoras, folios, etc.) y gastos de reprografía (fotocopias de documentación) derivados del mismo.

Así, el **coste total en material fungible** asciende a **191.90 €.**

A.6. Costes generales

En este apartado se incluyen todos los gastos generales que no es posible cuantificar directamente. Dentro de esta partida se imputan gastos del tipo: agua, luz, costes de las comunicaciones (teléfonos, postales, etc.) gas, amortización y mantenimiento de edificios, etc. Aunque estos gastos suelen suponer un coste importante dentro del presupuesto, no existe una forma prefijada para calcularlos.

Los costes generales de acuerdo con la normativa vigente de la UC3M se cuantifican en un 15% de los costes directos de personal y otros gastos. En el caso de este proyecto no se ha imputado nada a otros gastos, ya que todos los costes podían reflejarse en otros apartados. Por tanto la base para este cálculo son los gastos directos de personal.

De esta forma, los **costes generales** ascienden a **9237.23 €**.

A.7. Impuesto sobre el valor añadido

Como ocurre en cualquier proyecto, hay que incrementar el importe total obtenido en el presupuesto con el I.V.A. correspondiente a los tipos vigentes, es decir, un 18%, y que se incluirá en la facturación.

El cálculo del I.V.A. se realiza sobre la base imponible, esto es, la suma de los conceptos de costes directo por personal, costes de materiales inventariables, costes de materiales fungibles y costes generales.

Atendiendo a esta regla el **Impuesto sobre el Valor Añadido** para este proyecto asciende a **12881.94 €**.

A.8. Total

Por último, solo queda poner en orden todos los conceptos referidos en los apartados anteriores para obtener el importe total que figurará en el contrato:

CONCEPTO	IMPORTE (€)
Personal	61581.48
Material Inventariable	555.69
Material Fungible	191.90
Costes Generales	9237.23
Total (Base Imponible)	71566.30
I.V.A. (18%)	12881.94
TOTAL FACTURA:	**84448.24**

Tabla 20: Resumen del Presupuesto

El presupuesto total de este proyecto asciende a la cantidad de **84448.24 €.**

Madrid a 25 de Febrero de 2012

El Ingeniero Proyectista

ANEXO B: PLANIFICACIÓN Y DIAGRAMA DE GANTT

	Nombre	Duración	Inicio	Terminado	Predecesores
1	Inicio del Proyecto	1 day?	5/09/11 8:00	5/09/11 17:00	
2	**Fase de Planificación**	**14 days?**	**6/09/11 8:00**	**23/09/11 17:00**	**1**
3	Recopilación de Informes	5 days?	6/09/11 8:00	12/09/11 17:00	
4	Estudio de posibles soluciones	6 days?	13/09/11 8:00	20/09/11 17:00	3
5	Elección de la solución	3 days?	21/09/11 8:00	23/09/11 17:00	4
6	**Fase de Análisis y Diseño**	**30 days?**	**26/09/11 8:00**	**4/11/11 17:00**	**2**
7	Estudio de Requisitos Funcionales	5 days?	26/09/11 8:00	30/09/11 17:00	
8	Estudio de Requisitos No Funcionales	5 days?	3/10/11 8:00	7/10/11 17:00	7
9	Elección de la estructura del sistema	8 days?	10/10/11 8:00	19/10/11 17:00	8
10	Diseño Final de la Solución	12 days?	20/10/11 8:00	4/11/11 17:00	9
11	**Fase de Desarrollo**	**110 days?**	**7/11/11 8:00**	**6/04/12 17:00**	**6**
12	**Programación**	**110 days?**	**7/11/11 8:00**	**6/04/12 17:00**	
13	**Programación del Sistema**	**85 days?**	**7/11/11 8:00**	**2/03/12 17:00**	
14	Reunión Seguimiento Sistema 0	0,375 days?	7/11/11 8:00	7/11/11 11:00	
15	Desarrollo Estructuras Básicas	9,625 days?	7/11/11 11:00	18/11/11 17:00	14
16	Reunión Seguimiento Sistema 1	0,375 days?	21/11/11 8:00	21/11/11 11:00	15
17	Desarrollo Tablas de Confianza en Contenidos	9,625 days?	21/11/11 11:00	2/12/11 17:00	16
18	Reunión Seguimiento Sistema 2	0,375 days?	5/12/11 8:00	5/12/11 11:00	17
19	Desarrollo Tablas de Confianza en Peers	9,625 days?	5/12/11 11:00	16/12/11 17:00	18
20	Reunión Seguimiento Sistema 3	0,375 days?	19/12/11 8:00	19/12/11 11:00	19
21	Desarrollo Tablas de Riesgos	4,625 days?	19/12/11 11:00	23/12/11 17:00	20
22	Reunión Seguimiento Sistema 4	0,375 days?	26/12/11 8:00	26/12/11 11:00	21
23	Desarrollo Gestor de Tablas	14,625 days?	26/12/11 11:00	13/01/12 17:00	22
24	Reunión Seguimiento Sistema 5	0,375 days?	16/01/12 8:00	16/01/12 11:00	23
25	Desarrollo Gestor de Logs	4,625 days?	16/01/12 11:00	20/01/12 17:00	24
26	Reunión Seguimiento Sistema 6	0,375 days?	23/01/12 8:00	23/01/12 11:00	25
27	Desarrollo de las Interfaces	4,625 days?	23/01/12 11:00	27/01/12 17:00	26
28	Reunión Seguimiento Sistema 7	0,375 days?	30/01/12 8:00	30/01/12 11:00	27
29	Desarrollo Clase Peer	24,625 days?	30/01/12 11:00	2/03/12 17:00	28
30	**Programación de la Aplicación de Prueba**	**25 days?**	**5/03/12 8:00**	**6/04/12 17:00**	**13**
31	Reunión Seguimiento Aplicación 0	0,375 days?	5/03/12 8:00	5/03/12 11:00	
32	Desarrollo del Menú de Administrador	4,625 days?	5/03/12 11:00	9/03/12 17:00	31
33	Reunión Seguimiento Aplicación 1	0,375 days?	12/03/12 8:00	12/03/12 11:00	32
34	Desarrollo del Menu de Contenidos	9,625 days?	12/03/12 11:00	23/03/12 17:00	33
35	Reunión Seguimiento Aplicación 2	0,375 days?	26/03/12 8:00	26/03/12 11:00	34
36	Desarrollo del Menú de Peers	4,625 days?	26/03/12 11:00	30/03/12 17:00	35
37	Reunión Seguimiento Aplicación 3	0,375 days?	2/04/12 8:00	2/04/12 11:00	36
38	Desarrollo del Menú de Grupos	4,625 days?	2/04/12 11:00	6/04/12 17:00	37
39	**Fase de Pruebas**	**65 days?**	**9/04/12 8:00**	**6/07/12 17:00**	**11**
40	Reunión Seguimiento Pruebas 0	0,375 days?	9/04/12 8:00	9/04/12 11:00	
41	Prueba de Estructuras Básicas	4,625 days?	9/04/12 11:00	13/04/12 17:00	40
42	Reunión Seguimiento Pruebas 1	0,375 days?	16/04/12 8:00	16/04/12 11:00	41
43	Prueba de Tablas de Confianza	9,625 days?	16/04/12 11:00	27/04/12 17:00	42
44	Reunión Seguimiento Pruebas 2	0,375 days?	30/04/12 8:00	30/04/12 11:00	43
45	Prueba del Gestor de Tablas	9,625 days?	30/04/12 11:00	11/05/12 17:00	44
46	Reunión Seguimiento Pruebas 3	0,375 days?	14/05/12 8:00	14/05/12 11:00	45
47	Prueba del Gestor de Logs	4,625 days?	14/05/12 11:00	18/05/12 17:00	46
48	Reunión Seguimiento Pruebas 4	0,375 days?	21/05/12 8:00	21/05/12 11:00	47
49	Pruebas de la Clase Peer	19,625 days?	21/05/12 11:00	15/06/12 17:00	48
50	Reunión Seguimiento Pruebas 5	0,375 days?	18/06/12 8:00	18/06/12 11:00	49
51	Pruebas Aplicación de Prueba	14,625 days?	18/06/12 11:00	6/07/12 17:00	50
52	**Fase de Documentación**	**50 days?**	**9/07/12 8:00**	**14/09/12 17:00**	**39**
53	Redacción de la documentación	50 days?	9/07/12 8:00	14/09/12 17:00	
54	Fin de Proyecto	1 day?	17/09/12 8:00	17/09/12 17:00	52

Figura 62: Planificación del Proyecto

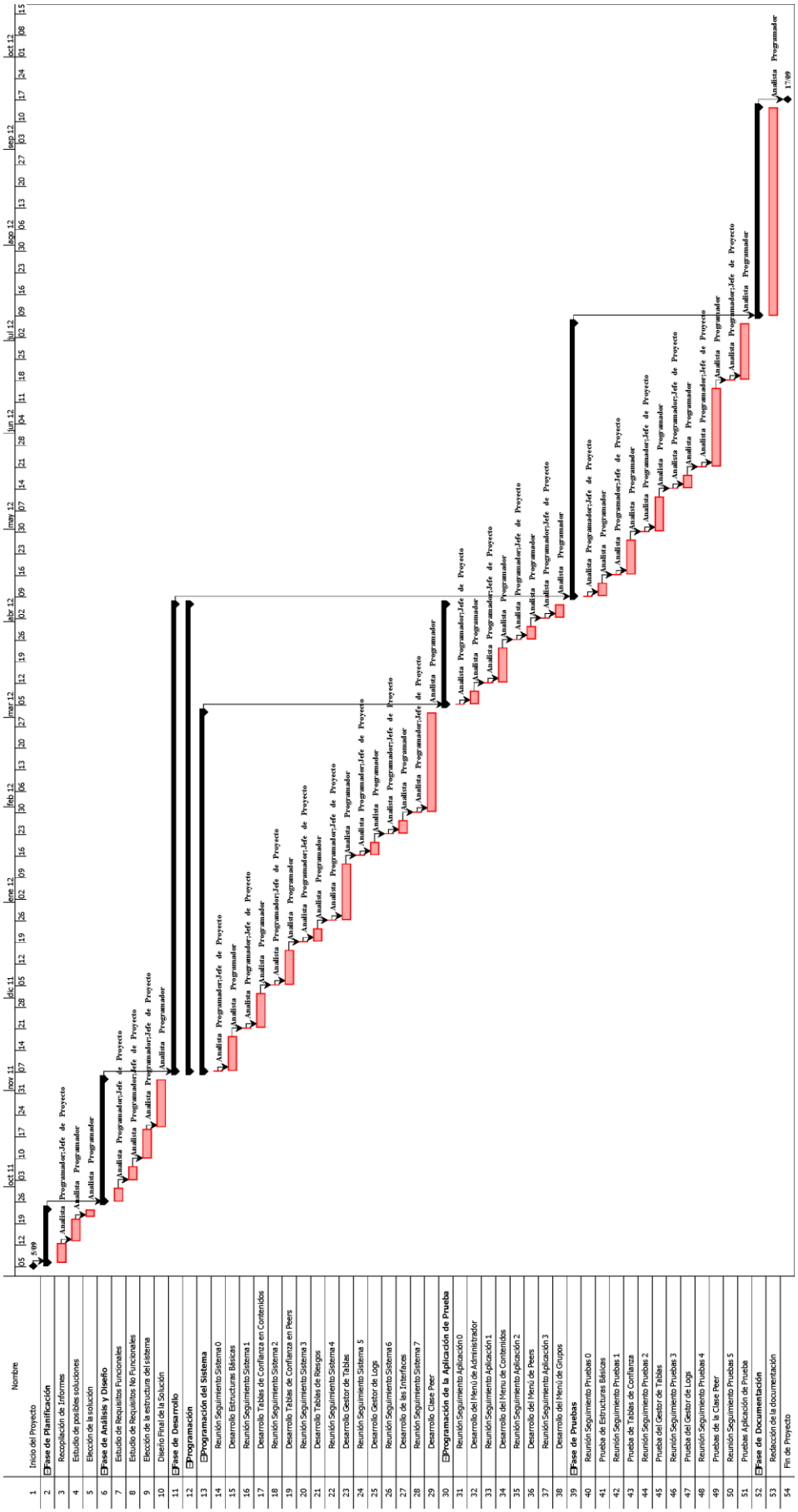


Figura 63: Diagrama de Gantt de la Planificación

ANEXO C: MANUAL DE LA APLICACIÓN DE PRUEBA

C.1. Introducción

El sistema de gestión de reputación descrito en esta memoria es completamente funcional por sí mismo, permitiendo que cualquier aplicación externa pueda utilizar sus interfaces para acceder a sus utilidades. Del mismo modo, este proyecto incluye una aplicación de prueba que permitirá a un usuario acceder a las mismas utilidades desde un terminal de comandos.

Este anexo servirá para confeccionar un manual que permita el uso de la aplicación de prueba, mostrando todas las posibilidades que ofrece y guiando al usuario por cada uno de sus elementos. Para que este tutorial sea más comprensible, se dividirá en varios apartados que cubran la explicación detallada de cada uno de los menús de la aplicación, cuya estructura general se puede ver en la figura 64.

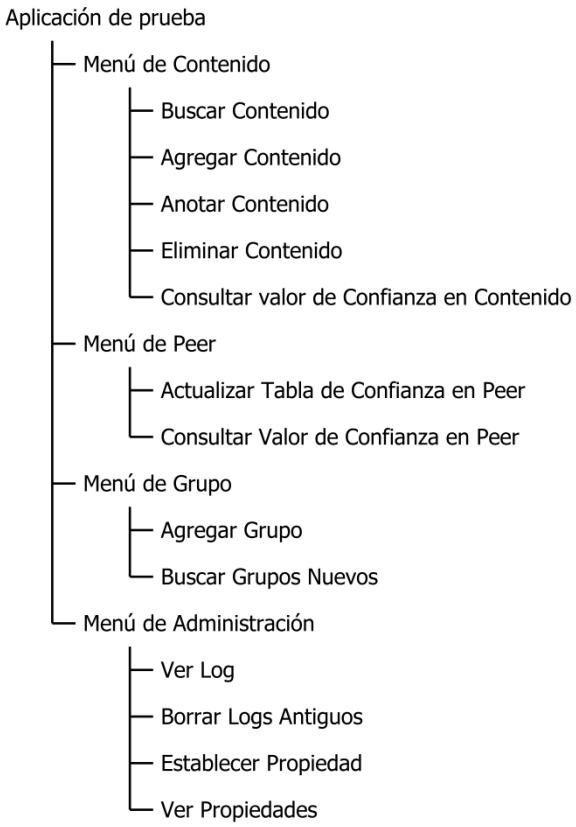


Figura 64: Estructura de la aplicación de prueba

C.2.Arranque de la aplicación

Antes de arrancar la aplicación, se debe hacer una presentación del contenido del directorio desde donde se ejecutará la misma. De esta forma se podrá ver si en dicha carpeta están todos los elementos necesarios para el correcto funcionamiento. El aspecto que presenta se puede ver en la figura 65.

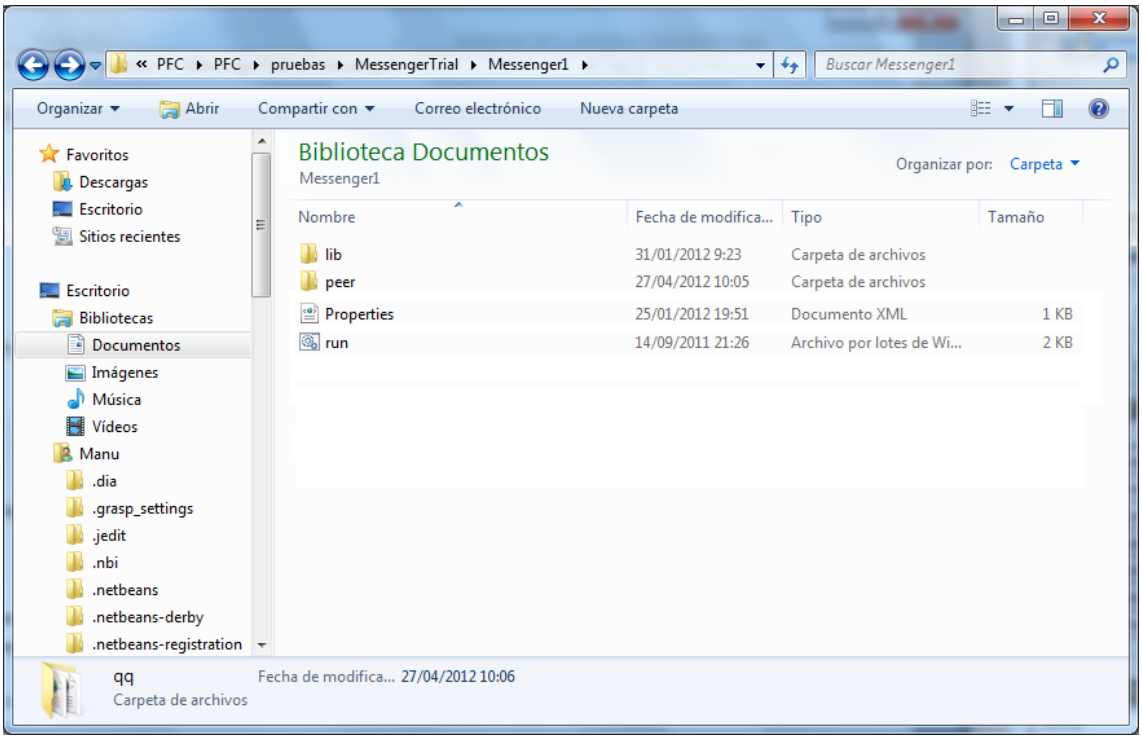


Figura 65: Directorio de la aplicación antes de la ejecución

Tal y como se puede observar, en el directorio existirán cuatro elementos que no pueden faltar, y que serán los siguientes:

- *lib*: Directorio que contiene todas las librerías de *JXTA* necesarias para la ejecución del programa.
- *peer*: Directorio que contiene los archivos del código fuente del sistema de gestión de reputación.
- *Properties*: Fichero de configuración de la aplicación.
- *run*: Script que compilará y lanzará la aplicación de prueba.

Para ejecutar la aplicación se debe abrir un terminal de comandos y navegar por los directorios hasta situarse en el mismo nivel que el directorio mencionado antes, donde se encuentra el *script* de lanzamiento. Una vez ahí se debe ejecutar la aplicación, tal y como muestra la figura 66.

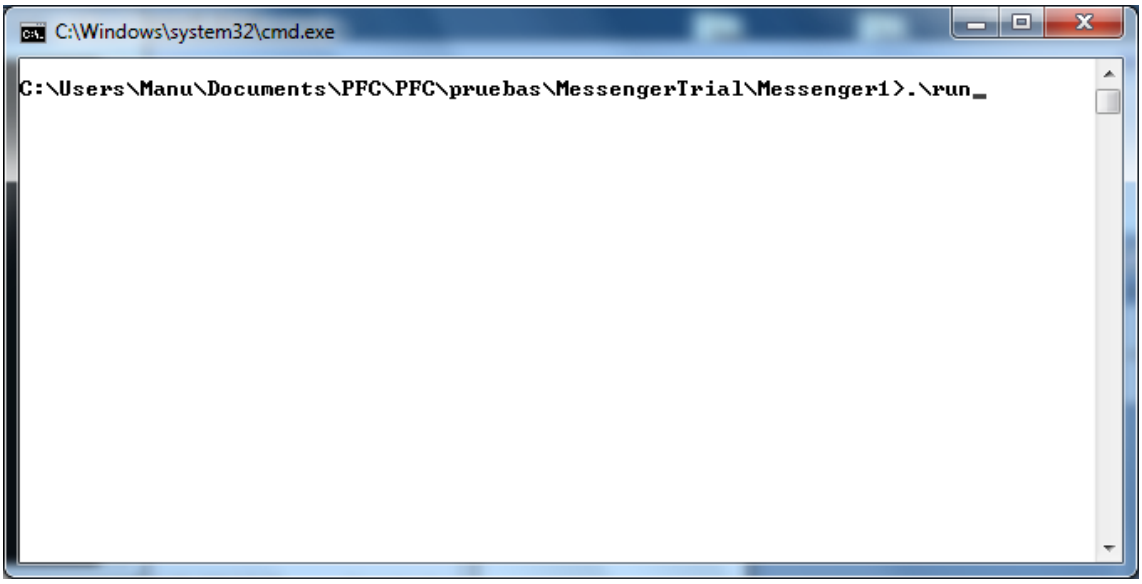


Figura 66: Lanzamiento de la aplicación

Una vez hecho esto, el script realizará su trabajo y se encargará de la compilación y la ejecución de la aplicación de prueba. Si todo es correcto, se le pedirá al usuario la pulsación de cualquier tecla y, a continuación, solicitará el nombre de usuario. Este proceso se puede observar en la figura 67.

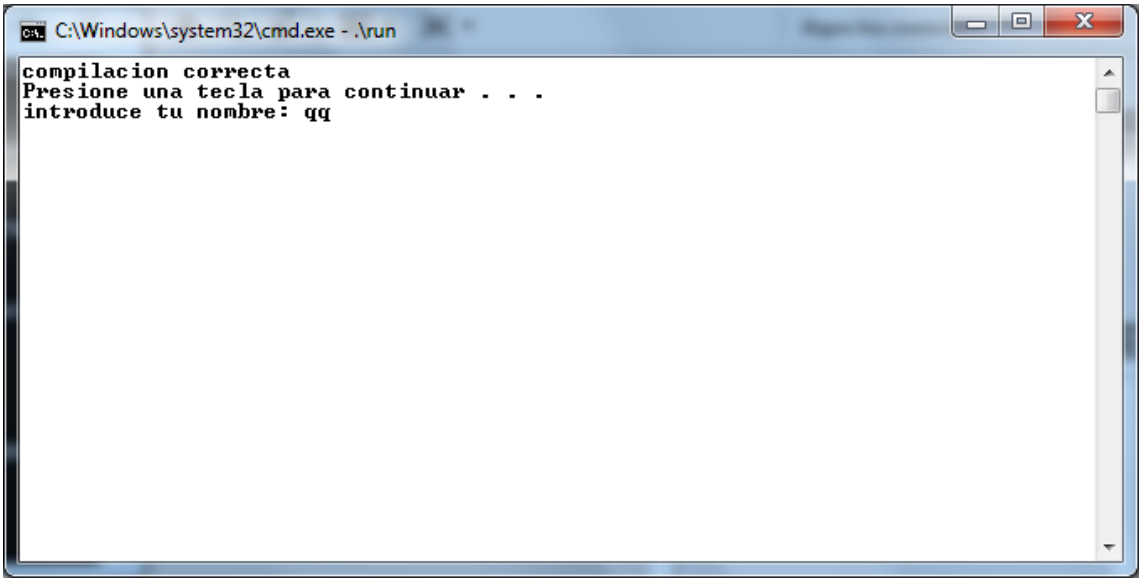


Figura 67: Introducción del nombre en la aplicación

En este momento, el usuario no tendrá que hacer nada más para que la aplicación termine de arrancar y muestre el menú principal. Si todo ha ido bien se mostrará una pantalla como la que se muestra en la figura 68, en la que se podrá ver el número de puerto obtenido por el *peer*, el nombre del grupo por defecto al que pertenecerá y el identificador *JXTA* único que se le ha asignado, y que será permanente para todas las ejecuciones del sistema.

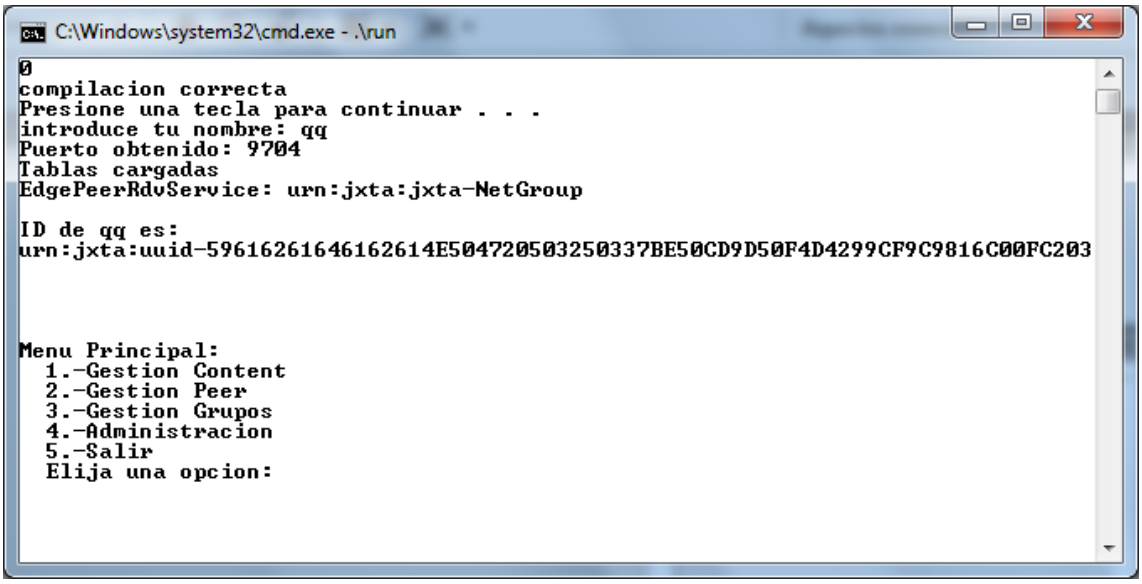


Figura 68: Arranque completo de la aplicación

Desde el punto de vista del directorio de ejecución de la aplicación, que se mostraba al comienzo de esta sección, se podrá observar un cambio fundamental: la creación de un nuevo directorio que será en el que se almacene toda la información referente al usuario, tal y como se puede ver en la figura 69.

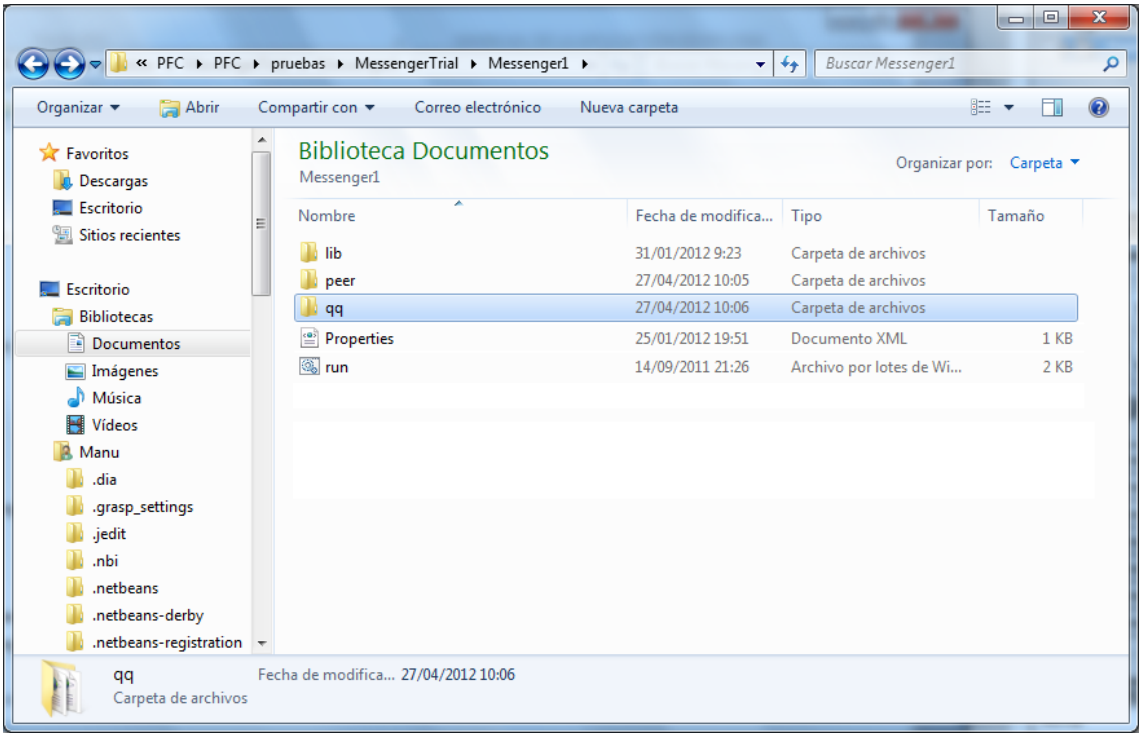


Figura 69: Directorio de la aplicación tras la ejecución

En esta nueva carpeta se almacenará toda la información relativa al usuario, desde la configuración básica de la plataforma *JXTA*, incluido el identificador que utilizará siempre, hasta los ficheros de *log*, pasando por las tablas que almacenarán toda la información de la confianza, clave para el funcionamiento del sistema. El contenido de este directorio puede verse en la figura 70.

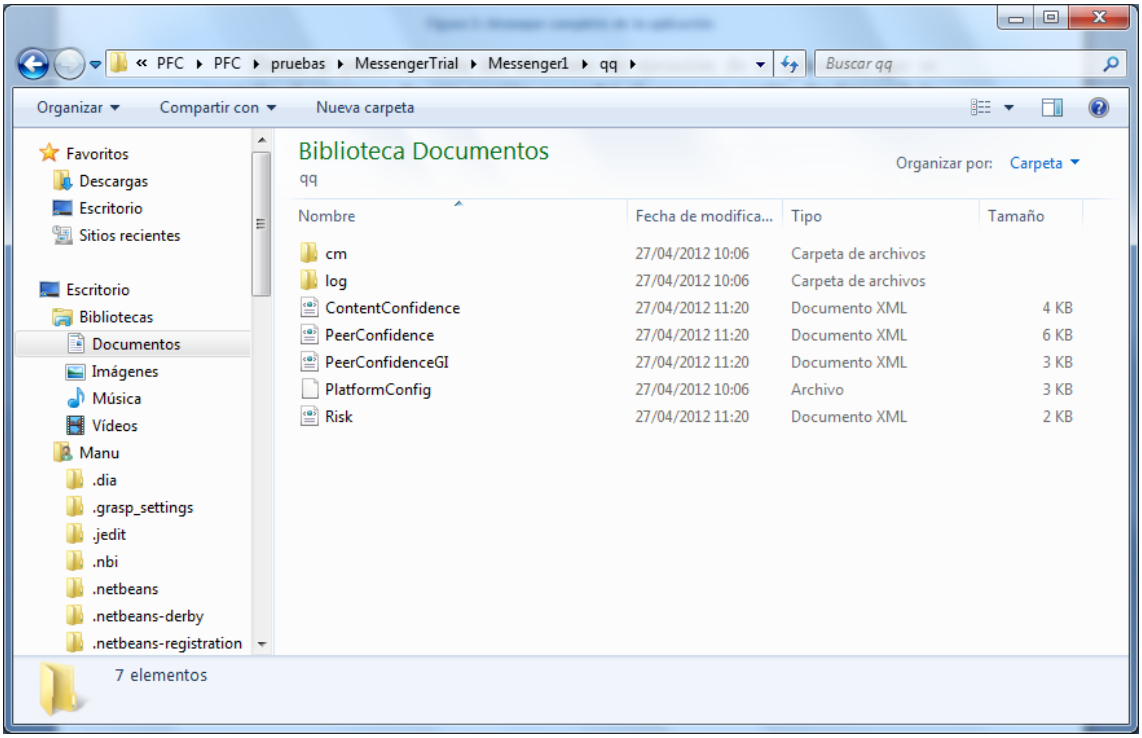


Figura 70: Directorio de usuario

Tal y como se puede observar, el directorio de usuario estará formado por los siguientes elementos:

- *cm*: Este directorio contendrá toda la información que necesita la plataforma *JXTA* para su correcto funcionamiento.
- *log*: Es el directorio que almacenará los ficheros de *log*.
- *ContentConfidence*, *PeerConfidence*, *PeerConfidenceGI* y *Risk*: Ficheros *XML* que servirán para el almacenamiento de las tablas de confianza.
- *PlatformConfig*: Fichero utilizado por la plataforma *JXTA* para guardar toda la información relativa a la identidad del usuario.

Una vez descrito el proceso de arranque de la aplicación de prueba se puede pasar a describir, en las siguientes secciones, el contenido y la funcionalidad de cada uno de los menús de la aplicación.

C.3.Menú principal

El menú principal de la aplicación constituye el primer contacto del usuario con la aplicación de prueba y, por lo tanto, con el sistema de gestión de reputación. Este menú servirá para dar acceso al usuario a los menús específicos que permitirán gestionar el sistema, tal y como se puede ver en la figura 71.

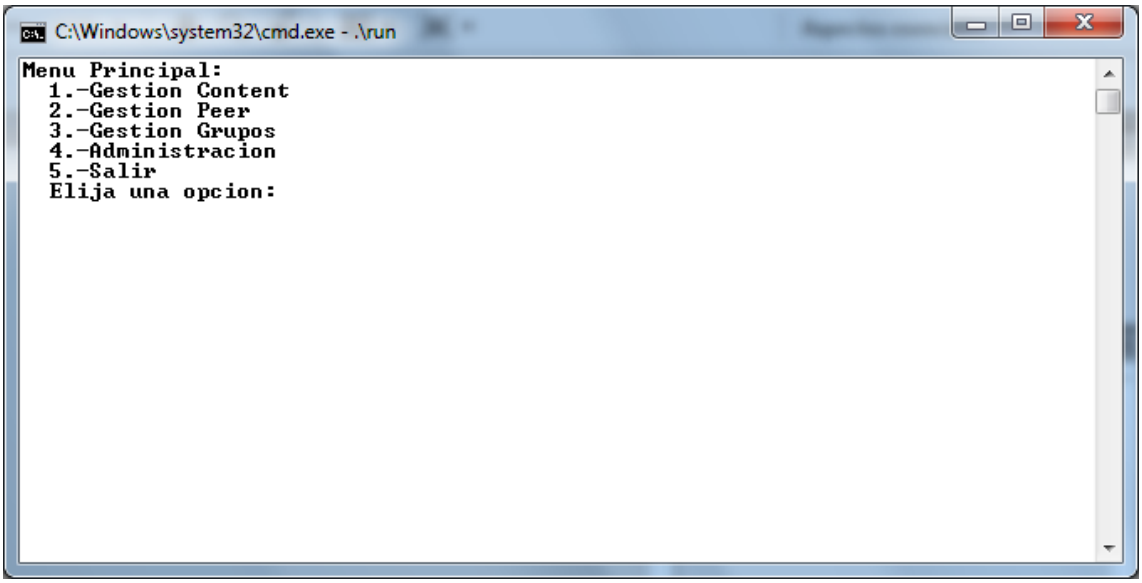


Figura 71: Menú principal de la aplicación

Aunque la captura anterior resulta bastante significativa, y a pesar de que los diferentes submenús se detallarán en las secciones siguientes, se puede realizar una breve reseña a cada una de las opciones que da este menú:

- **Gestión Content**: Permite al usuario acceder al menú que se encarga de la gestión de los contenidos dentro del sistema de gestión de la reputación.
- **Gestión Peer**: Da acceso al menú que se encarga de la gestión de los *peers* dentro del sistema de gestión de la reputación.
- **Gestión Grupos**: Permite acceder al menú encargado de gestionar los grupos a los que pertenece el *peer* dentro de la red.
- **Administración**: Permite al usuario acceder al menú de gestión de la configuración del sistema.
- **Salir**: Permite al usuario salir de la aplicación de prueba.

C.4.Menú de contenido

C.4.1. Introducción

Este menú permite al usuario acceder al conjunto de opciones que ofrece la aplicación de prueba para la gestión de los contenidos, que se compartirán y evaluarán en el sistema de gestión de reputación. El contenido de este menú se puede observar en la figura 72.

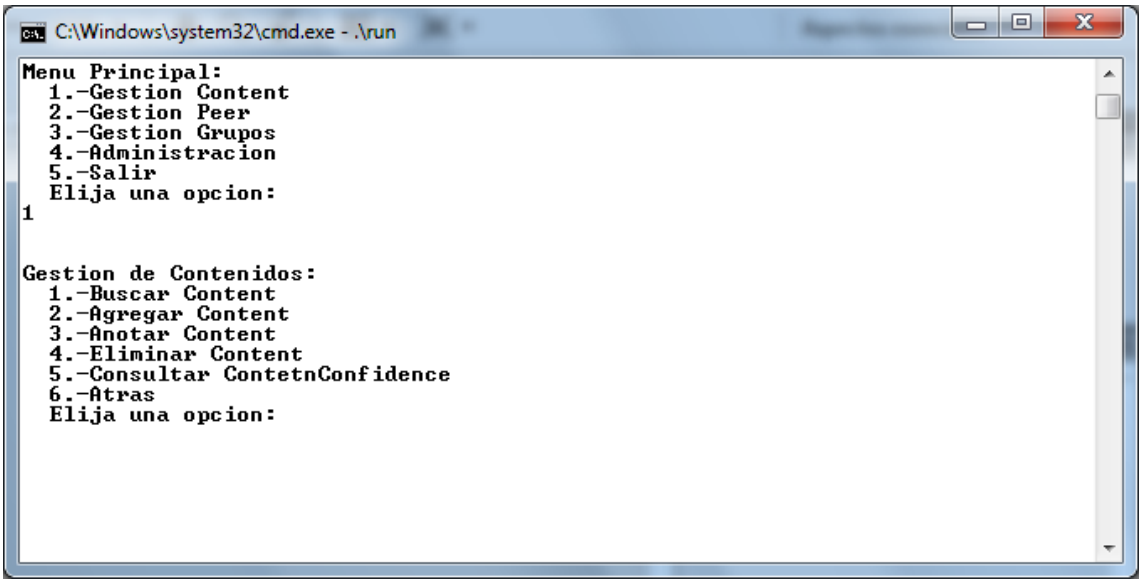


Figura 72: Menú de contenidos

A continuación, en los siguientes apartados, se mostrará la utilidad y el funcionamiento de cada una de las opciones de este menú, detallando con las correspondientes capturas su comportamiento.

C.4.2. Buscar content

Esta opción permitirá al usuario tener acceso a la funcionalidad más importante del sistema de gestión de reputación: la búsqueda de contenidos que se ajusten a sus preferencias. Este es procedimiento más complejo de todos los que se pueden encontrar en la aplicación de prueba ya que, además de realizar la búsqueda del contenido, se deberá realizar la evaluación del mismo.

En primer lugar, al seleccionar esta opción, el sistema solicitará al usuario la palabra clave a la que hace referencia el contenido que se está buscando. Esta palabra podrá ser cualquier cadena de caracteres, aunque por lo general serán palabras sencillas que faciliten la búsqueda.

Una vez introducida la palabra clave, la aplicación mostrará un listado con todos grupos de intereses disponibles a los que el usuario podrá dirigir la búsqueda. El usuario deberá elegir uno de ellos para comenzar el proceso de búsqueda.

Como último parámetro se deberá especificar si la búsqueda desea realizarse en remoto o si, por el contrario, primero se desea consultar las tablas locales de contenidos. El usuario deberá introducir su decisión con los caracteres "S" o "N".

Tras un breve lapso de tiempo, el sistema se encargará de efectuar la búsqueda y mostrar por pantalla los resultados obtenidos, en caso de que los hubiese. Esta parte del proceso de búsqueda puede verse en la figura 73.

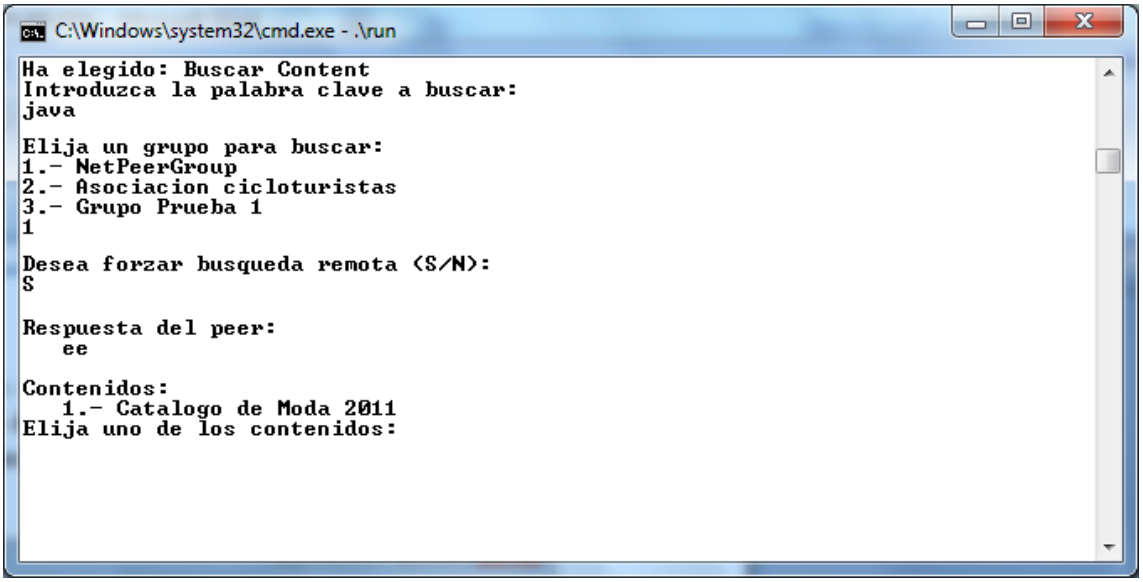


Figura 73: Primera fase de la búsqueda de un contenido

En el caso particular del ejemplo anterior, el listado de resultados encontrados para la búsqueda contiene un único elemento pero, por lo general, habrá más de un contenido entre los que poder elegir. Una vez que el usuario elija, a través de la introducción del índice correspondiente, uno de los contenidos, se realizará el acceso al mismo, aunque en este caso la lectura será simulada.

Tras este proceso, el sistema solicitará una puntuación que refleje el grado de satisfacción del usuario con dicho contenido, y que deberá estar comprendida entre -1 y 4. Una vez introducido este valor, el sistema comenzará con el proceso de evaluación del contenido, que tendrá dos fases diferenciadas, aunque para el usuario serán transparentes.

Por un lado, el sistema actualizará las tablas locales de confianza en contenido, actualizando el valor previo que tuviese dicho recurso, en caso de que lo hubiese, o añadiendo la nueva entrada. Por otro lado, el sistema enviará automáticamente un *feedback* con la nueva puntuación al *peer* remoto del que se recibieron los contenidos.

Cada uno de estos pasos tiene su correspondiente resultado por pantalla, con el objetivo de mantener informado en todo momento al usuario acerca de lo que está pasando en la aplicación, tal y como puede verse en la figura 74.

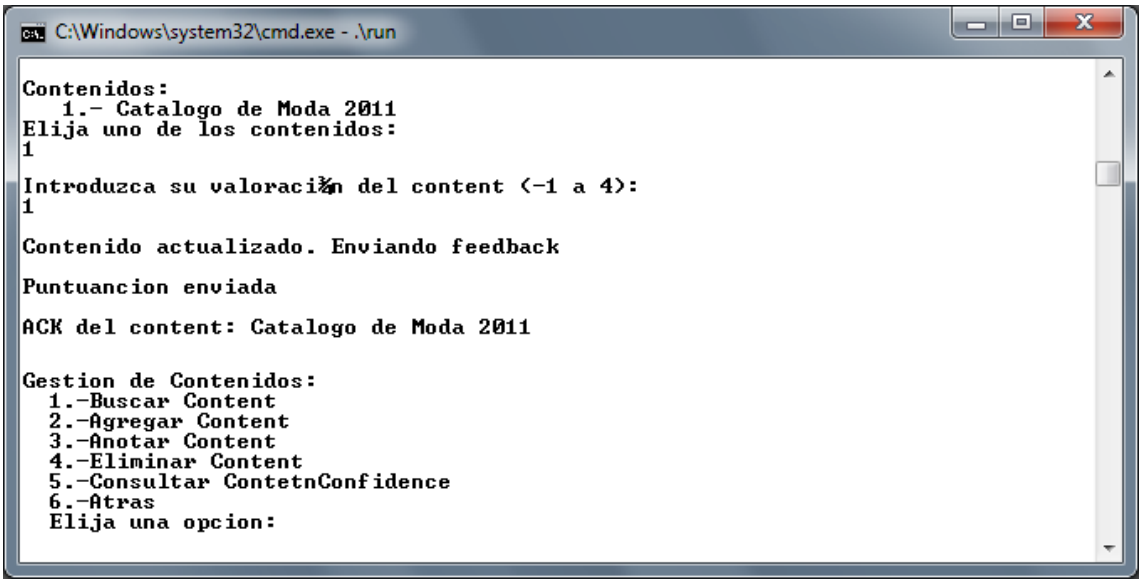


Figura 74: Evaluación de un contenido

Como se puede observar en la figura anterior, al finalizar el proceso de evaluación se muestra por pantalla un mensaje que indica que se ha recibido un *ACK* (*Acknowledgment*), que indicará que el nodo remoto que proporcionó los resultados actualizó correctamente sus tablas de confianza en base al *feedback* enviado.

Una vez concluido el proceso de búsqueda y evaluación de un contenido, la aplicación de prueba volverá a mostrar el menú de contenidos, para que el usuario pueda continuar trabajando con la misma.

C.4.3. Agregar content

Una característica interesante de la aplicación de prueba es la posibilidad de crear nuevos contenidos y agregarlos como nuevas entradas en las tablas locales de confianza. Esto permitirá a *peers* remotos encontrar dichos contenidos para su evaluación.

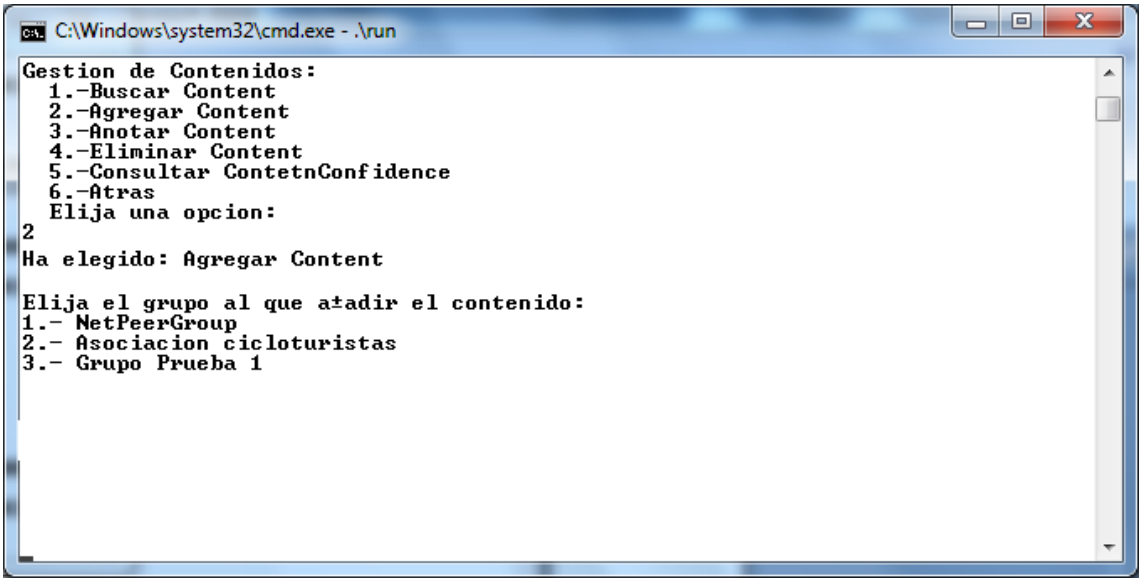


Figura 75: Primer paso para agregar un contenido

El proceso de creación de un contenido es bastante sencillo, y comienza con la petición al usuario del grupo en el que se desea añadir dicho contenido nuevo, para ubicarlo correctamente dentro de las tablas locales de confianza. Esto se puede observar en la figura 75.

Una vez elegido el grupo de intereses, el sistema solicitará al usuario la palabra clave a la que hace referencia. Del mismo modo, esta palabra clave servirá para que el contenido se almacene correctamente dentro de las tablas de confianza.

Por último, el sistema solicitará el nombre del contenido, que servirá para que la representación del mismo en el futuro, tanto en el sistema local como en otros nodos de la red, sea más amigable. El proceso se puede ver en la figura 76.

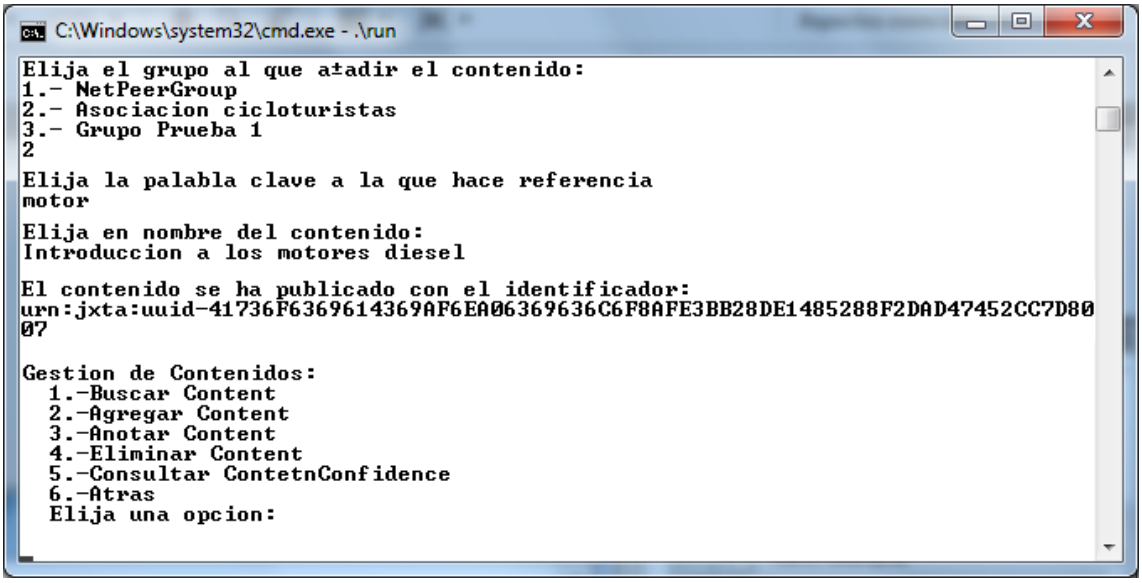


Figura 76: Proceso completo para agregar un contenido

Tal y como se puede ver en la captura anterior, el sistema devolverá un identificador *JXTA* único con el que se relacionará dicho contenido a partir de ese momento. Por último, la aplicación de prueba volverá a mostrar el menú de gestión de contenidos.

C.4.4. Etiquetar content

En cualquier momento, un usuario puede observar que un contenido determinado hace referencia a una palabra clave concreta, pero que las tablas locales no representan dicha relación. Cuando esto ocurre, el sistema incluye los mecanismos necesarios para que el usuario añada dicha palabra clave al contenido.

El primer paso para etiquetar un contenido es solicitar el nombre del contenido concreto para el que se desea añadir una nueva palabra clave. En este punto se debe decir que el usuario deberá introducir el nombre exacto con el que dicho contenido fue dado de alta en el sistema y almacenado en las tablas de confianza. Seguiremos con el ejemplo del contenido creado en el apartado anterior, como se puede ver en la figura 77.

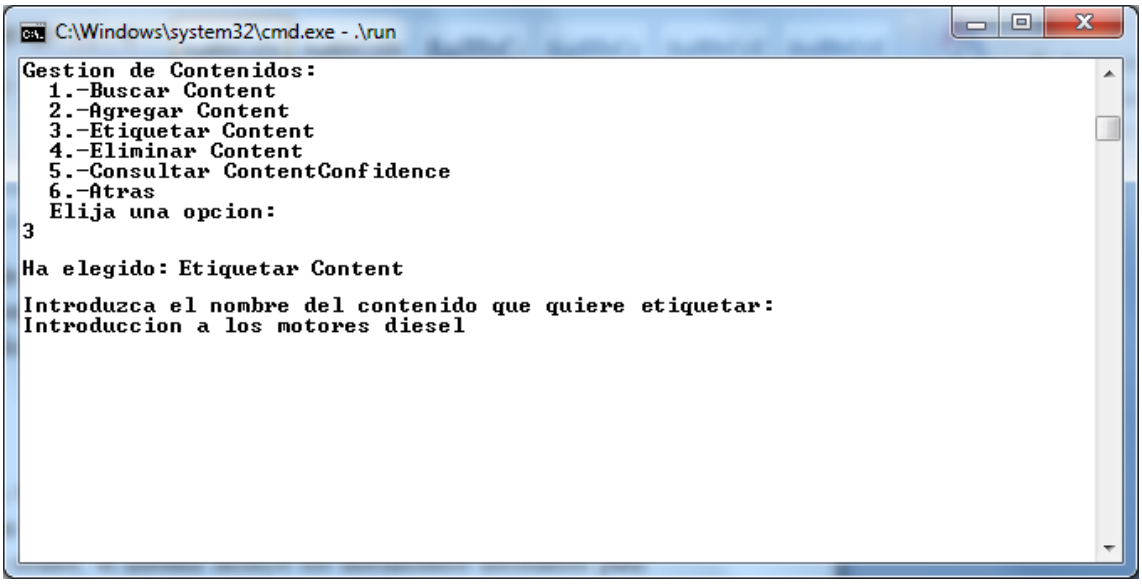


Figura 77: Primera paso para etiquetar un contenido

El segundo y último paso para anotar un contenido será introducir la nueva palabra clave a la que hace referencia. En este momento, la aplicación de prueba solicitará al usuario que introduzca dicha palabra, lo que ocasionará que todas las tablas de los grupos en los que exista dicho contenido, se añadirán nuevas entradas referentes a las nuevas palabras clave.

Tras este paso se volverá a mostrar el menú de gestión de contenidos, tal y como puede verse en la figura 78.

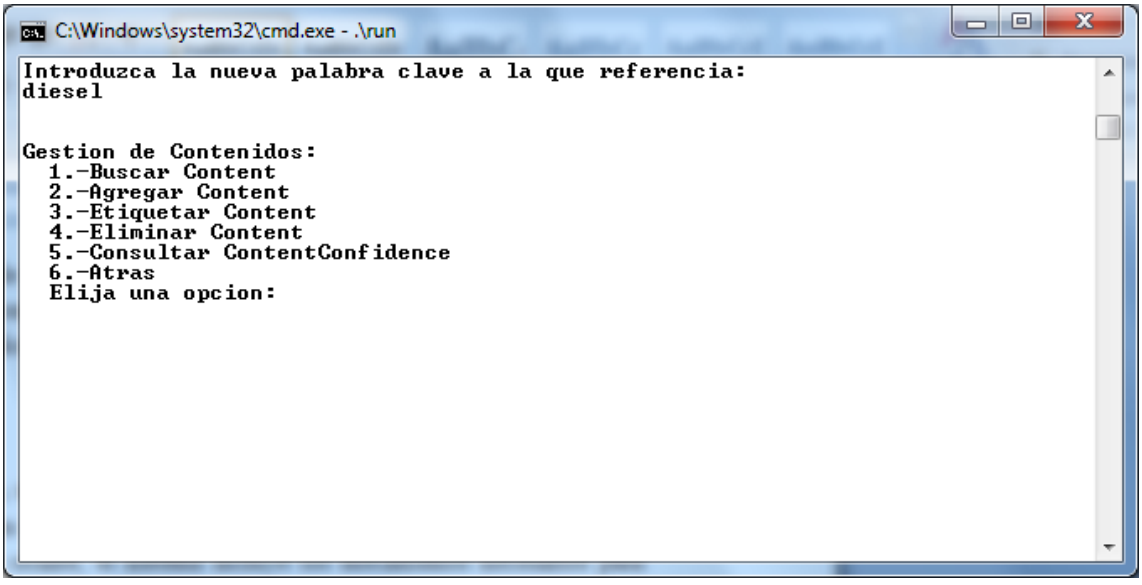


Figura 78: Segundo paso para etiquetar un contenido

C.4.5. Eliminar content

De forma complementaria a la opción descrita en el apartado anterior, la aplicación de prueba ofrece al usuario la posibilidad de eliminar de las tablas de confianza locales un determinado contenido. Esto significa que, desde el punto de vista del usuario, dicho contenido no hace referencia a una palabra clave determinada, por lo que habría que eliminar la correspondiente entrada de las tablas.

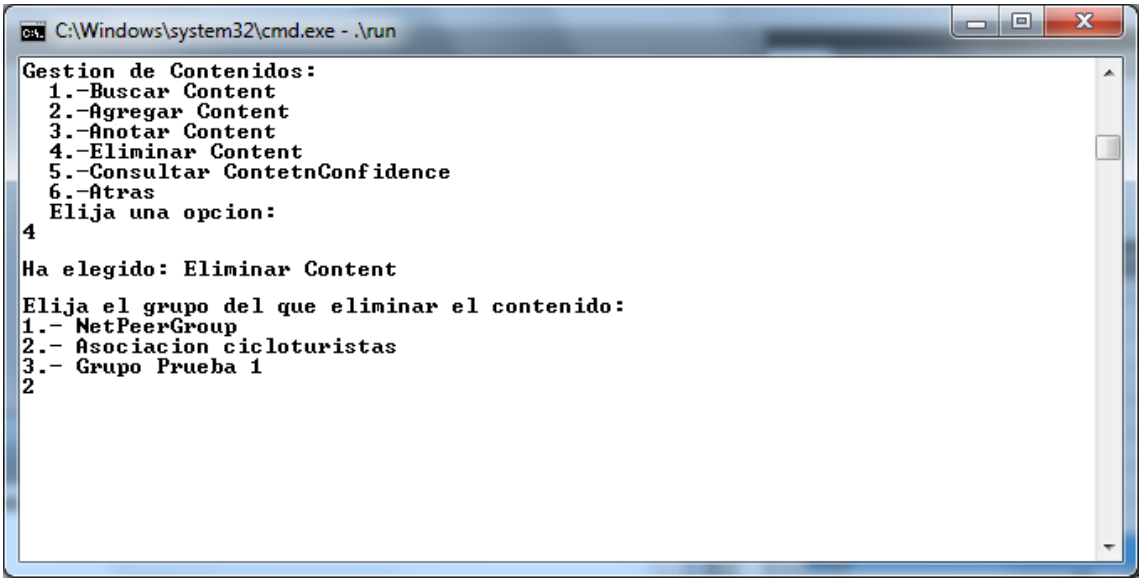


Figura 79: Comienzo de la eliminación de un contenido

En primer lugar, como se puede ver en la figura 79, la aplicación pedirá al usuario que elija el grupo del que se desea eliminar dicho contenido. El motivo de este paso previo a la eliminación es doble: por una parte servirá para concretar el grupo del que eliminar el contenido, ya que el propio usuario puede querer que el mismo deje de estar presente en dicho grupo; por otra parte servirá para buscar el contenido dentro de las tablas.

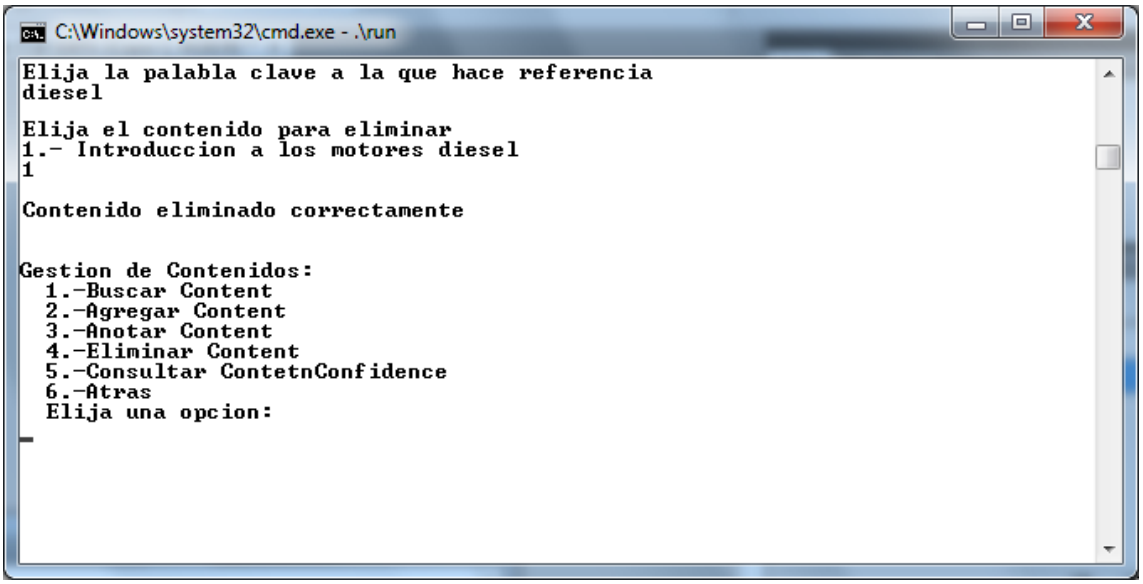


Figura 80: Eliminación de un contenido completada

El segundo paso para hacer efectiva la búsqueda del contenido que será eliminado, es solicitar al usuario la palabra clave a la que hace referencia. Esto servirá para concretar la búsqueda de dicho contenido y, si efectivamente concuerda con uno o más contenidos almacenados en las tablas locales, se mostrará un listado con los recursos disponibles para borrar, tal y como se muestra en la figura 80.

En este momento, el usuario deberá seleccionar el contenido que desea eliminar y, si todo el proceso se efectúa correctamente, se mostrará un mensaje que lo confirme, volviendo de nuevo al menú de gestión de contenido.

C.4.6. Consultar confianza en contenido

La última de las posibilidades que ofrece el menú de gestión de contenidos es la consulta de un valor de confianza para un contenido concreto. Esto permitirá al usuario conocer la valoración que tiene un contenido dentro del sistema local.

Como ocurría en otros casos, para poder consultar un valor concreto se debe especificar tanto el grupo en el que se desea realizar la consulta, como el nombre exacto de dicho contenido, tal y como se puede ver en la figura 81. En el caso del nombre, se introducirá directamente por el usuario, mientras que el grupo se elegirá de un listado de los disponibles en las tablas locales.

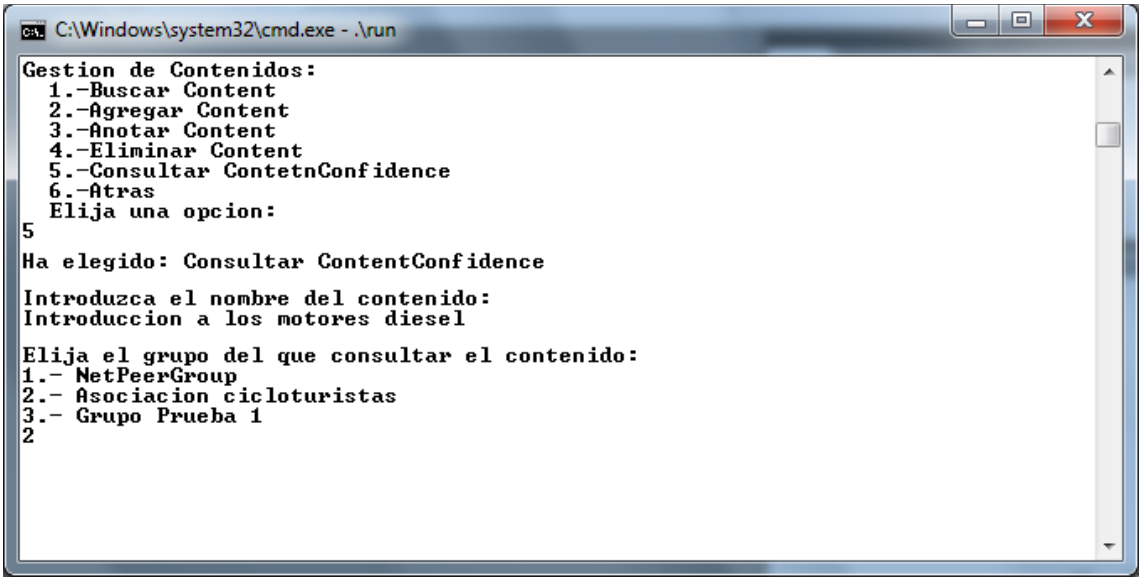


Figura 81: Primera fase de consulta de contenido

Por último, el usuario deberá especificar la palabra clave a la que hace referencia el contenido, ya que un mismo contenido puede tener diferentes valoraciones en función de la palabra clave que se esté teniendo en cuenta. Al igual que ocurre con el nombre del contenido, el usuario deberá introducir directamente por teclado la palabra clave.

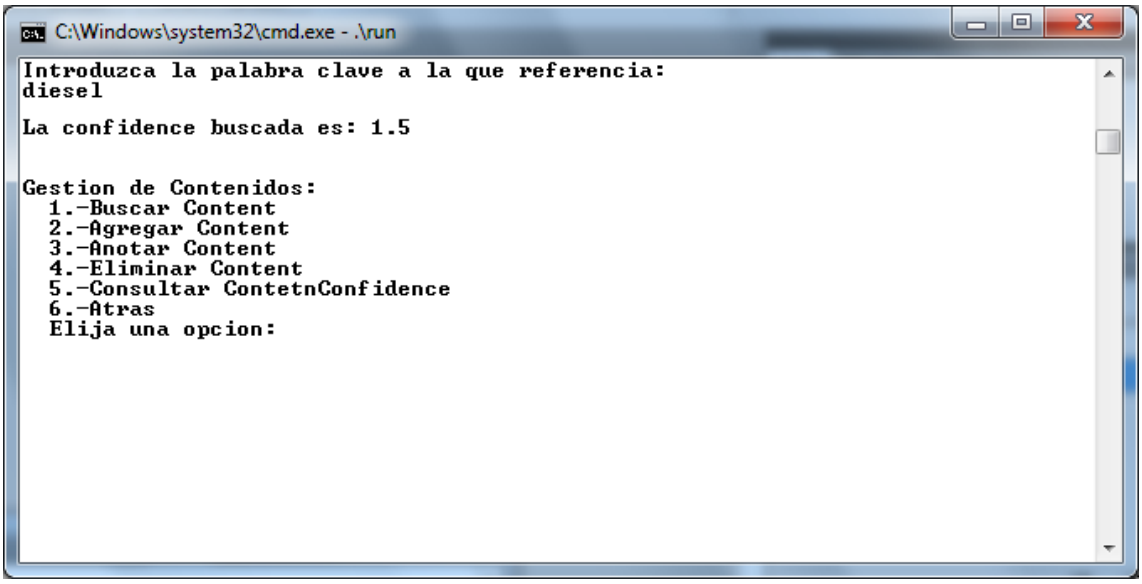


Figura 82: Fin de la consulta de un valor de confianza

Una vez determinado unívocamente el valor de confianza que se desea consultar, con los parámetros grupo, palabra clave y nombre, el sistema mostrará el valor almacenado, tal y como puede verse en la figura 82. En caso de no encontrarse ningún valor que concuerde con la consulta, se mostrará un mensaje de error, que indicará que la búsqueda no ha sido exitosa. Este final puede verse en la figura 83.

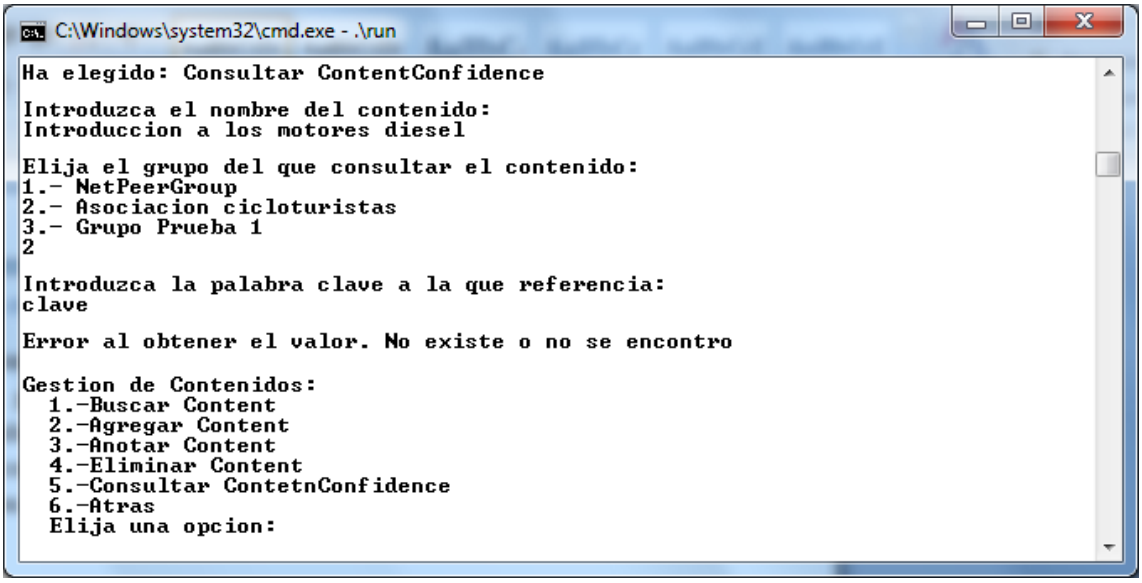


Figura 83: Consulta de confianza en contenido fallida

C.5.Menú de peer

C.5.1. Introducción

Este menú permite acceder al usuario al conjunto de opciones que ofrece la aplicación de prueba en lo que a la gestión de *peers* se refiere. El aspecto de este menú puede observarse en la figura 84.

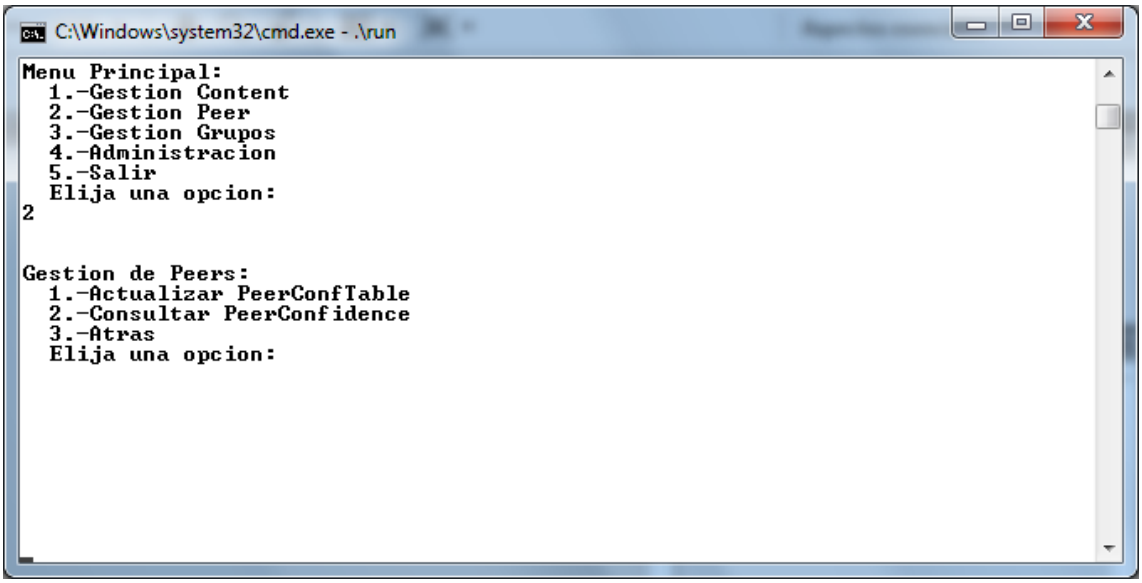


Figura 84: Menú de peer

En los siguientes apartados se describirá con detalle el modo de funcionamiento de cada una de las opciones incluidas en este menú que, como se puede observar, es mucho más sencillo que el correspondiente al de gestión de los contenidos.

C.5.2. Actualizar tabla de confianza

La primera de las opciones que nos da este menú consiste en la actualización de las tablas locales de confianza en *peer*. Esta opción permite ampliar la información contenida en las tablas locales con los datos almacenados en nodos remotos.

Como ocurría con otras opciones manejadas por el menú de gestión de contenidos, el primer paso para ejecutar esta opción consiste en indicarle al sistema qué grupo es el que se desea actualizar. Esta elección se efectúa sobre un listado de los grupos disponibles, del que el usuario deberá seleccionar el índice deseado.

Una vez el grupo ha sido elegido, automáticamente el sistema se encarga de consultar los *peers* conocidos en dichos grupos y enviar a cada uno de ellos las peticiones correspondientes para la actualización de las tablas de confianza. Estos nodos remotos enviarán la información de los *peers* a los que se podrá consultar en futuras búsquedas remotas de contenidos.

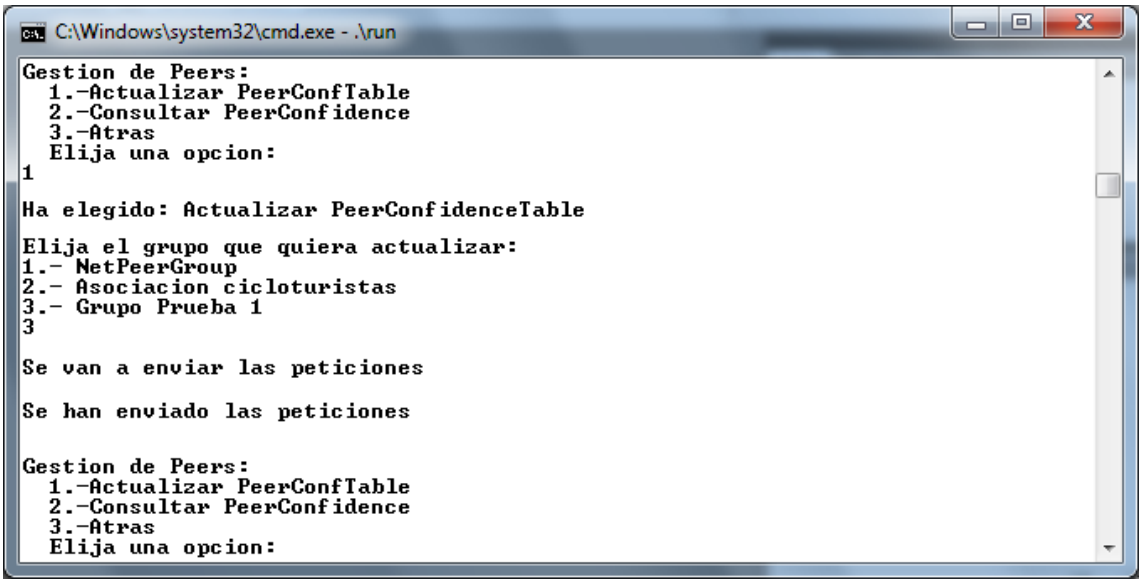


Figura 85: Actualización de la tabla de confianza en peer

Esta opción no muestra información acerca de los nuevos valores recibidos de los nodos remotos, sino que directamente estos datos pasarán a formar parte de las tablas locales para realizar nuevas búsquedas, ampliando de esta forma las probabilidades de encontrar contenidos. Por el contrario, lo que sí muestra esta opción es la confirmación de que se han podido enviar peticiones a nodos remotos, tal y como se muestra en la figura 85.

C.5.3. Consultar confianza en peer

Al igual que ocurría en el caso del menú de gestión de contenidos, el menú de gestión de *peers* debe incluir una opción que permita al usuario conocer el valor de confianza que el sistema local tiene para un nodo remoto. Esta opción necesitará que el usuario especifique el nombre del *peer* cuyo valor de confianza se quiere consultar, y la palabra clave para la que se quiere hacer la consulta, aunque se pueden dar dos posibilidades.

La primera de estas posibilidades es que el usuario quiera consultar dicho valor para un grupo específico. En este caso, tras la petición del nombre del nodo a consultar, se mostrará un listado con los grupos disponibles en los que el usuario podrá realizar la consulta. En este momento el usuario deberá introducir el índice del grupo deseado, tras lo cual se le solicitará la palabra clave sobre la que realizar la consulta.

Si todo el proceso de consulta se efectúa correctamente, y efectivamente hay un valor almacenado en las tablas locales que se corresponda con los parámetros especificados por el usuario, se mostrará un mensaje por pantalla que indicará la valoración para dicho nodo remoto en el rango de valores (-1,5). Un ejemplo de esta búsqueda se puede ver en la figura 86.

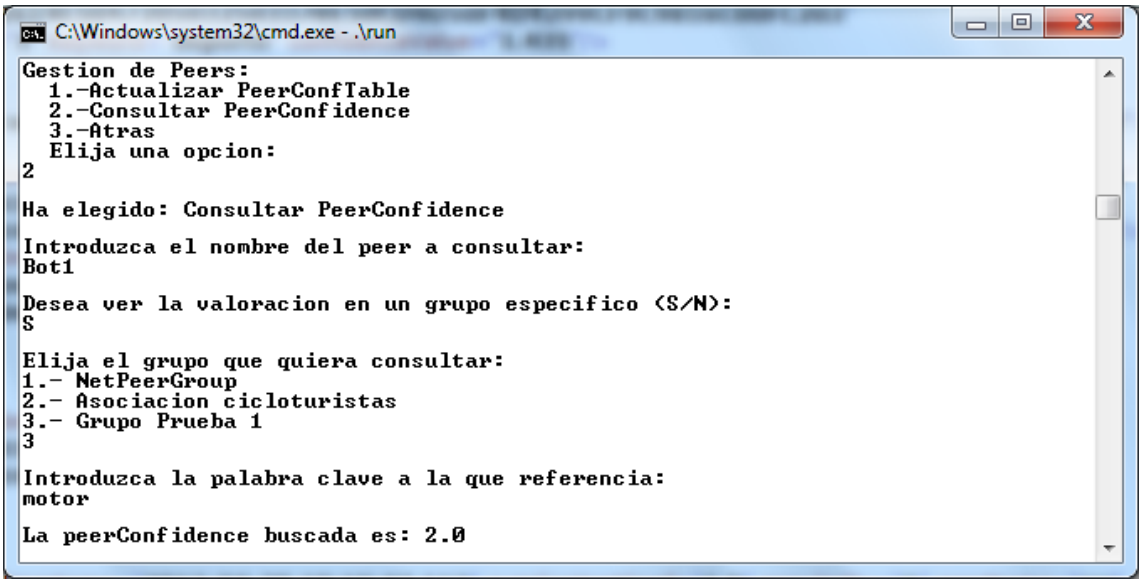


Figura 86: Consulta de confianza en peer en grupo específico

La segunda de las posibilidades que se mencionaban antes se da cuando el usuario no desea conocer el valor de confianza para un grupo determinado. En este caso, la consulta estará dirigida a todos los grupos disponibles en las tablas locales para las que se tenga un valor de confianza para el *peer* concreto. En este caso, a la pregunta de si se desea ver la valoración para un grupo específico el usuario deberá contestar que no.

Al igual que ocurría antes, el usuario deberá introducir la palabra clave sobre la que quiere realizar la consulta y, en este caso, la búsqueda se realizará sobre las tablas de confianza en *peer* independientes de grupo, que serán las que contengan la información buscada. Este proceso puede verse en la figura 87.

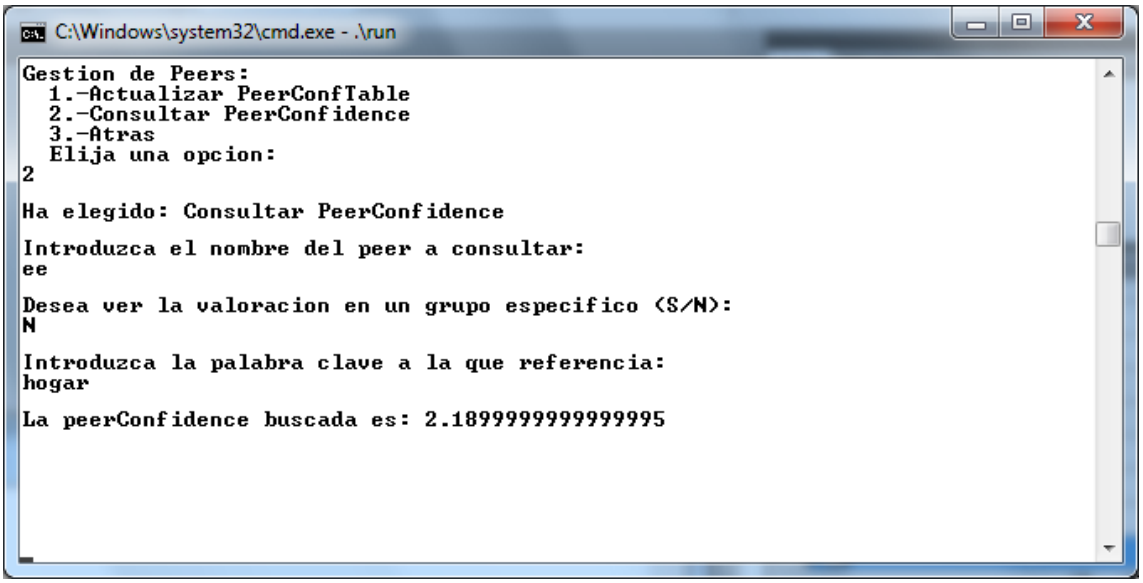


Figura 87: Consulta de confianza en peer sin grupo específico

Como ocurría en el caso de la consulta de valores de confianza en contenido, tanto para el caso de la consulta en un grupo específico como en la general, puede darse el caso de que no exista ningún resultado en las tablas locales para dicha búsqueda. En este caso, la aplicación de prueba mostrará un mensaje de error indicando que no se ha encontrado la información solicitada. Esto puede verse en la figura 88.

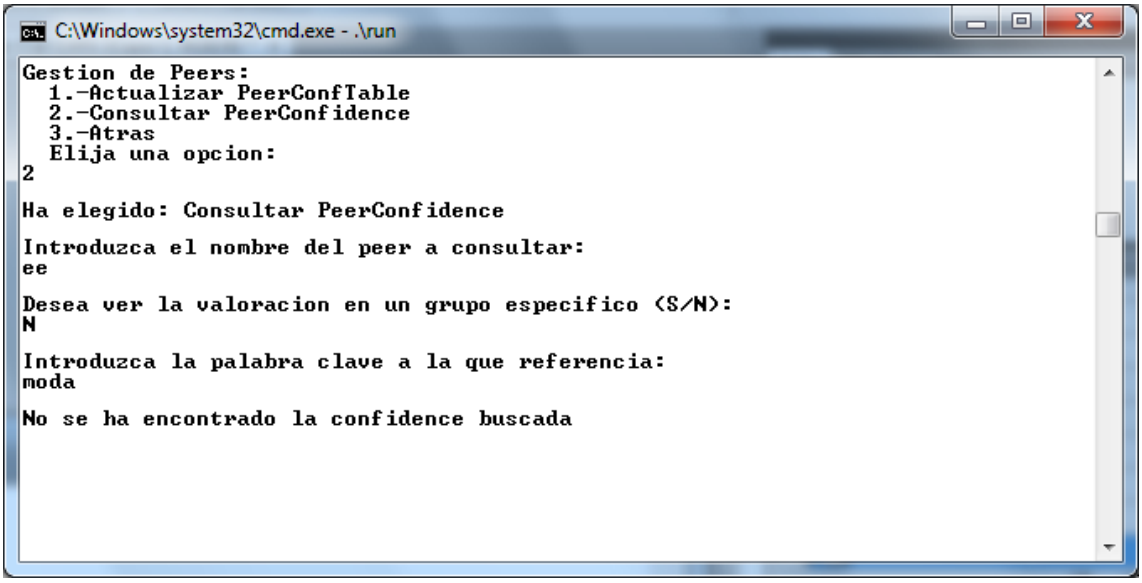


Figura 88: Consulta de confianza en peer fallida

C.6.Menú de grupo

C.6.1. Introducción

Este menú permite acceder al usuario al conjunto de opciones que ofrece la aplicación de prueba en lo que a la gestión de grupos se refiere. El aspecto de este menú puede observarse en la figura 89.

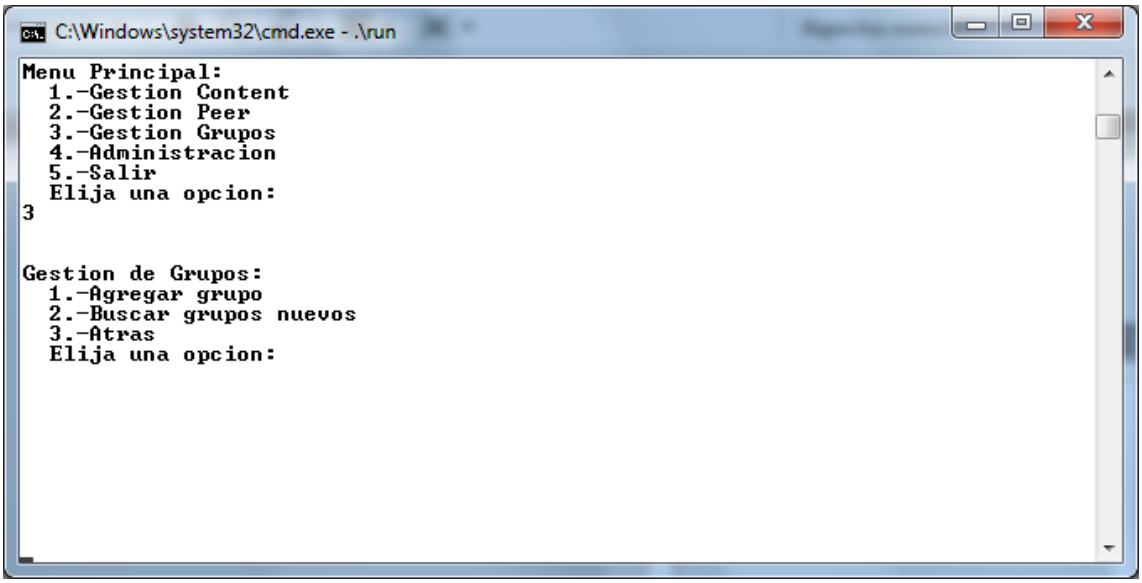


Figura 89: Menú de grupo

Los dos apartados siguientes servirán para detallar el funcionamiento de las opciones disponibles en este menú.

C.6.2. Agregar grupo

Cuando un usuario utiliza la aplicación de prueba, puede observar en un determinado momento, que no existe ningún grupo de intereses en la red que se ajuste a sus necesidades. En este caso, dicho usuario tendrá la opción de crear un nuevo grupo y publicarlo en la red para que otros nodos remotos puedan unirse a él.

El procedimiento para la creación de un grupo es muy sencillo, y el usuario estará guiado en todo momento por la aplicación de prueba. En primer lugar se deberá proporcionar un nombre al nuevo grupo, que será el utilizado a lo largo de toda su vida para que el sistema pueda mostrar la información correspondiente a los usuarios o aplicaciones que lo utilicen.

El segundo de los pasos consiste en dar una descripción más o menos detallada de la motivación del grupo, que servirá para que los nodos remotos que encuentren este nuevo grupo, sepan exactamente si se ajusta a su búsqueda. La primera parte de este proceso puede verse con detalle en la figura 90.

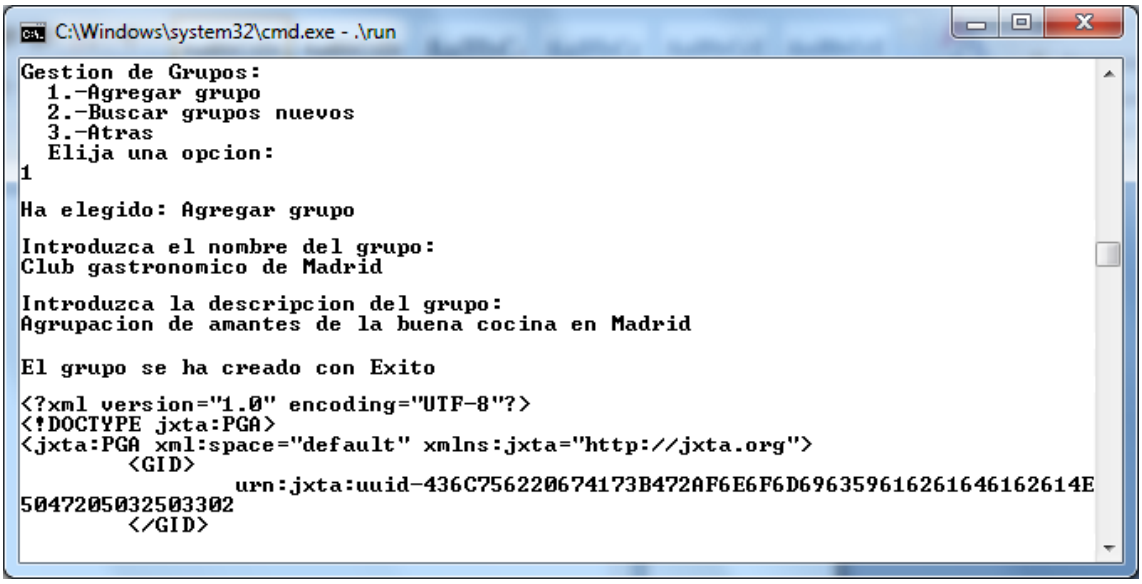


Figura 90: Primera fase para crear un grupo

Una vez que la aplicación tiene tanto el nombre como la descripción del grupo, el sistema será capaz de crear el grupo y, si todo sale bien, el usuario recibirá por pantalla un mensaje que refleje el éxito de la operación, así como el anuncio que será publicado en la red para que otros nodos puedan encontrarlo.

Finalmente, el usuario obtendrá por pantalla un mensaje que le indicará el identificador único *JXTA* que se le ha asignado al nuevo grupo dentro de la red, tal y como puede observarse en la figura 91.

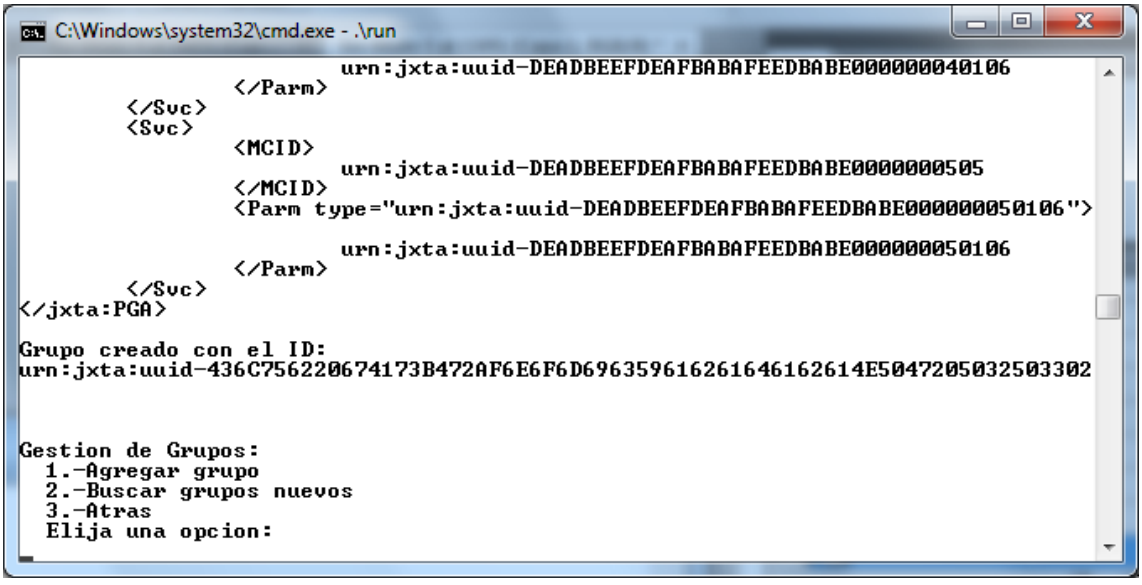


Figura 91: Mensaje final de la creación de un grupo

C.6.3. Buscar grupos nuevos

Antes de recurrir a la creación de un nuevo grupo de intereses, como se ha visto en el apartado anterior, el usuario tiene la posibilidad de buscar nuevos grupos en la red para comprobar si alguno de ellos coincide con sus requisitos y así evitar duplicidades.

En este caso, el usuario no tendrá que proporcionar ningún tipo de información cuando elija esta opción, sino que el sistema automáticamente se encargará de realizar la búsqueda en la red y mostrar los resultados obtenidos, en caso de que los hubiese.

Como muestra de esta opción, se puede seguir con el ejemplo del apartado anterior. Un *peer* habrá creado un determinado grupo y otro usuario realizará una búsqueda ejecutando la opción correspondiente. La salida por pantalla que se le ofrece se puede ver en la figura 92.

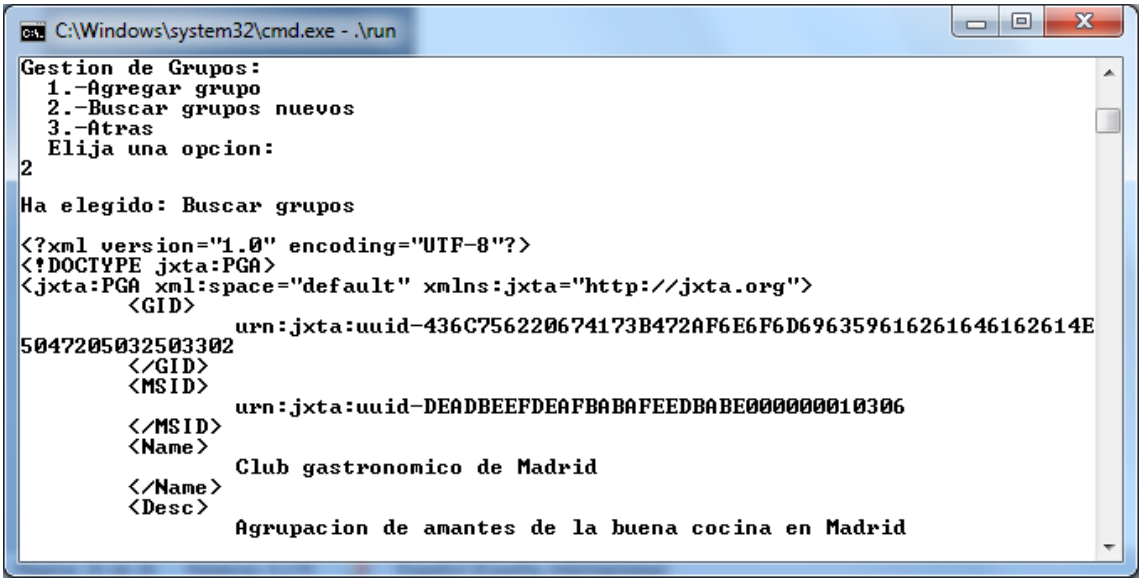


Figura 92: Comienzo de una búsqueda de grupos

Como se puede observar en esta captura, una vez que el usuario ejecuta la opción de búsqueda, se presentará por pantalla un listado con los anuncios completos de grupo que se han encontrado en la red. En principio, esta salida por pantalla contiene más información de la que el usuario necesita, pero en cada uno de los anuncios se podrá ver la descripción del grupo encontrado, permitiendo conocer al usuario si se ajusta a sus necesidades.

Siguiendo con la ejecución de la opción de búsqueda de nuevos grupos en la red, tras el listado de los anuncios de grupo que se han encontrado, aparecerá un resumen con los nombres de dichos grupos. Esto ayudará al usuario a conocer de un solo vistazo el conjunto de grupos existentes, como se puede observar en la figura 93.

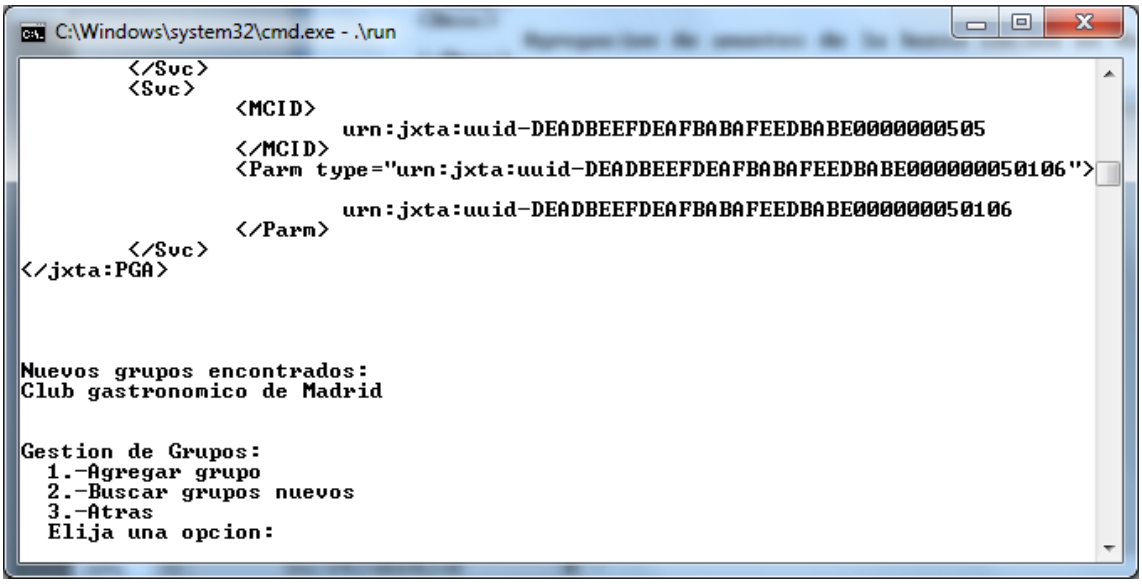


Figura 93: Fin de la búsqueda de grupos

Como se puede ver en la captura anterior, al finalizar el proceso de esta opción, se volverá a mostrar el menú de gestión de grupos, para que el usuario continúe su trabajo con la aplicación de prueba.

Tras la búsqueda de grupos, el usuario podrá buscar contenidos o actualizar la tabla de confianza en *peer* de los grupos encontrados, tal y como se mostró en las secciones correspondientes a los menús de gestión de *peer* y de contenidos.

C.7.Menú de administración

C.7.1. Introducción

El conjunto de menús visto hasta el momento en las secciones anteriores forma la base de operación de la aplicación de prueba, y muestra la funcionalidad que ofrece el sistema de gestión de la reputación a cualquier usuario o a cualquier aplicación externa.

De forma complementaria, además de proporcionar acceso a la funcionalidad del sistema, la aplicación de prueba también ofrece al usuario la posibilidad de realizar tareas de tipo administrativo, que se englobarán en un menú que se puede ver en la figura 94.

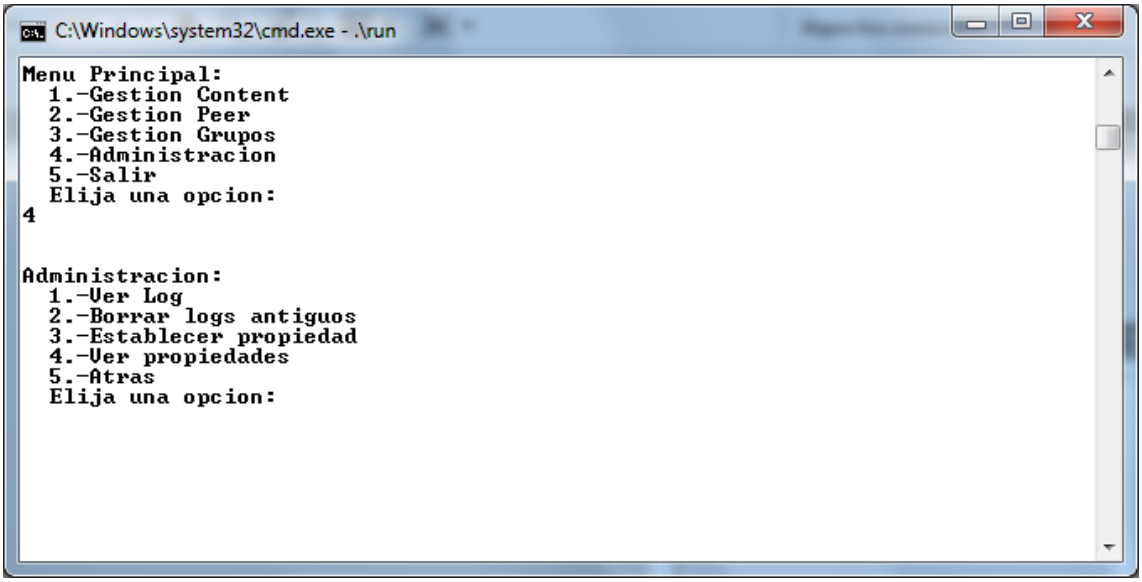


Figura 94: Menú de administración

C.7.2. Ver log

A lo largo del ciclo de vida del sistema, las continuas interacciones del *peer* local con otros nodos de la red estarán continuamente monitorizadas por el mismo, creando un conjunto de registros que se podrá consultar en cualquier momento.

Este registro servirá para verificar el correcto funcionamiento del sistema o, simplemente, para consultar la evolución temporal de los valores de confianza almacenados localmente, ya que en cada entrada de estos ficheros de *log* quedará reflejada tanto la operación llevada a cabo, como los nodos que intervinieron en ella, sin olvidar una marca temporal que permitirá ordenar cronológicamente todos los eventos.

En este caso, al ser una simple operación de consulta, el usuario no tendrá que proporcionar ningún parámetro de entrada para ejecutar su funcionalidad ya que, por defecto, se mostrará el contenido del último fichero de *log* creado, es decir, aquel fichero en el que se está escribiendo actualmente, mostrando así los últimos intercambios de información llevados a cabo. El comienzo del proceso puede verse en la figura 95.

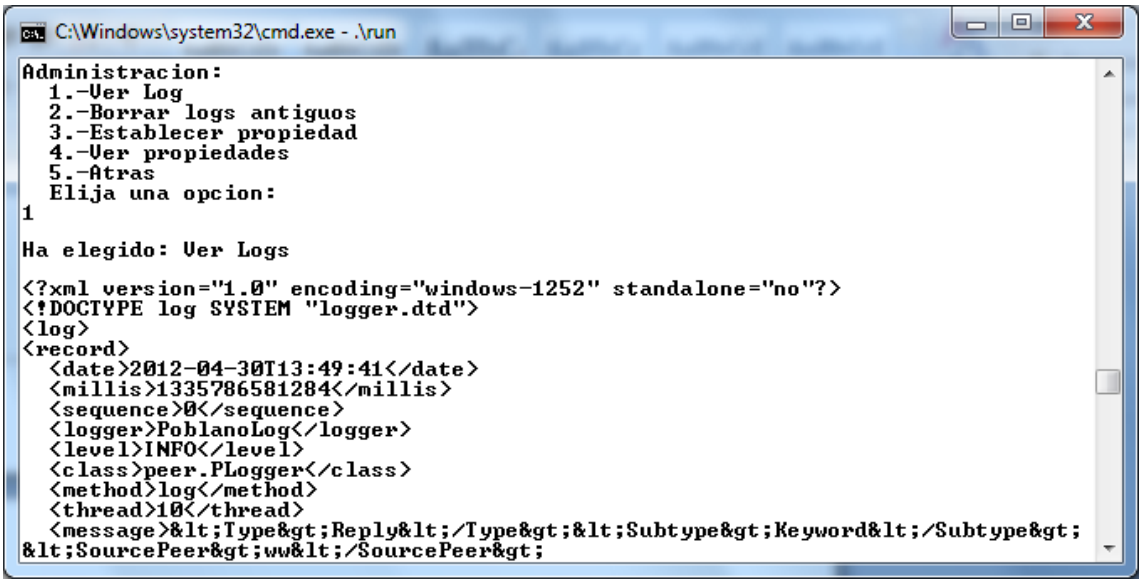


Figura 95: Comienzo de la visualización de log

Como se puede observar, cada entrada de este fichero es una estructura *XML* un tanto compleja, pero contiene toda la información necesaria para registrar la actividad de la aplicación. El registro continuará tal y como se muestra en la figura 96.

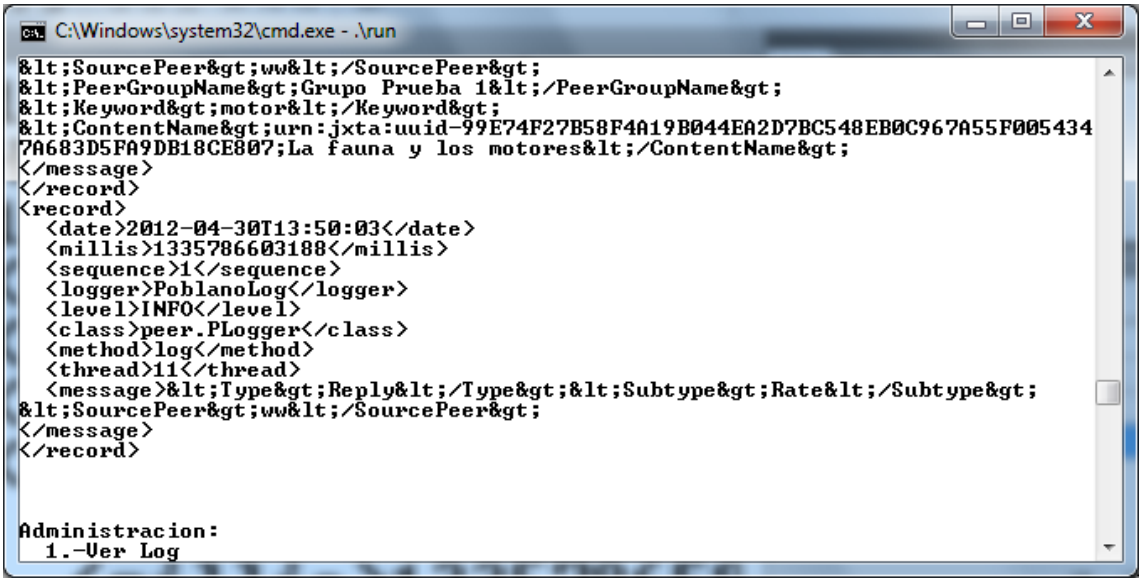


Figura 96: Fin de la visualización de log

Al finalizar la muestra del fichero de *log*, que puede ser más o menos extensa, dependiendo del tiempo que lleve en ejecución el sistema o el volumen de transacciones realizadas, se volverá a mostrar el menú de administración para que el usuario siga trabajando con la aplicación de prueba.

C.7.3. Borrar log antiguos

Cada vez que se ejecuta el sistema, se genera un fichero de *log* nuevo en el que se irán guardando los registros de la actividad del mismo, por lo que, en principio, habrá tantos ficheros como veces se haya ejecutado el sistema.

A pesar de que se trata de ficheros relativamente pequeños, es posible que el usuario desee liberar espacio de almacenamiento eliminando ficheros de *log* antiguos. Por este motivo, en la aplicación de prueba se incluye esta opción, que elimina todos los ficheros existentes exceptuando aquel que se corresponde con la ejecución actual del sistema. El proceso de esta opción se puede ver en la figura 97.

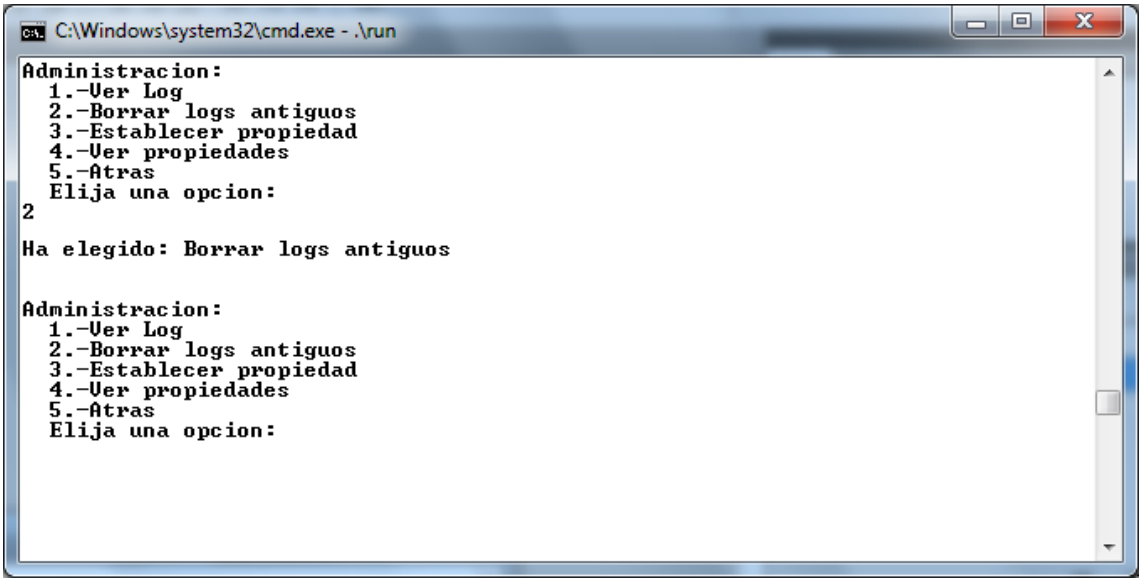


Figura 97: Borrado de ficheros de log antiguos

Al finalizar el borrado, la aplicación no dará ningún mensaje, lo que indicará que la operación se ha realizado con éxito. Sin embargo, la aplicación volverá a mostrar el menú de administración para que el usuario pueda continuar con su trabajo.

C.7.4. Ver propiedades

El sistema de gestión de reputación cuenta con varios parámetros que pueden ser configurados desde la aplicación de prueba. Entre estos parámetros podemos encontrar desde la ubicación del directorio de usuario hasta el intervalo entre guardados consecutivos de las tablas de confianza.

En cualquier momento de la ejecución del sistema, el usuario debería tener la opción de consultar los valores de estas propiedades, por lo que la aplicación de prueba contará con una opción para ello dentro del menú de administración.

Esta opción se encarga de consultar el fichero correspondiente a las propiedades del sistema, que estará situado en el directorio desde el que se ejecuta el mismo. Dicho fichero tendrá una estructura *XML* pero para facilitar la comprensión al usuario, se transformará, tal y como se puede ver en la figura 98.

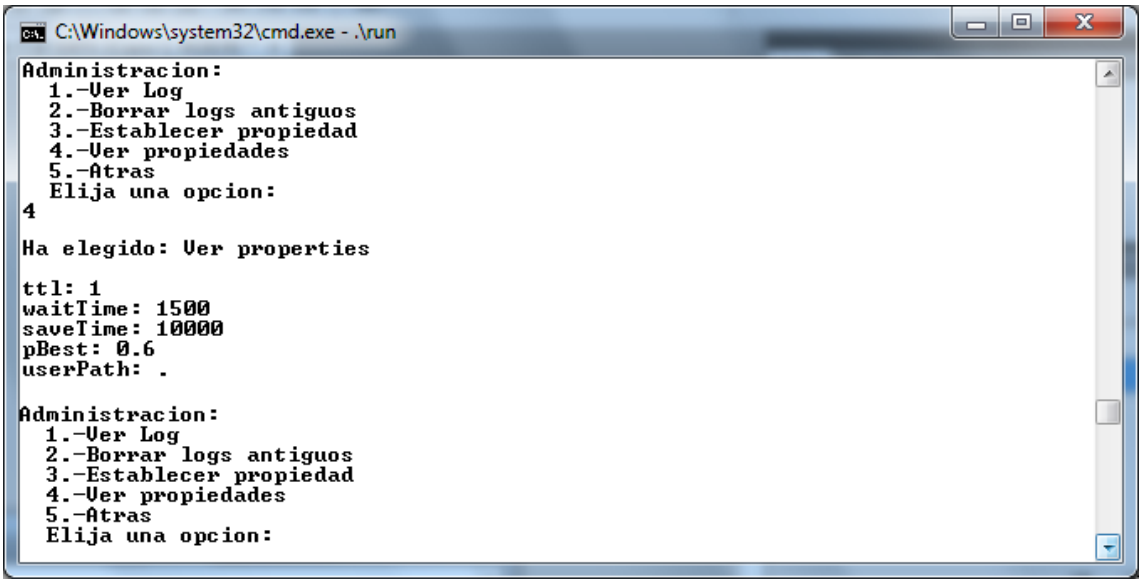


Figura 98: Visualización de propiedades

C.7.5. Establecer propiedad

La última de las opciones contenidas en la aplicación de prueba tiene que ver también con las propiedades de configuración del sistema, pero en este caso con su modificación. En cualquier momento, un usuario puede tener la necesidad de cambiar el valor de alguna de ellas, bien para comprobar los efectos que tiene sobre el sistema, o bien porque detecte que no presenta los valores adecuados.

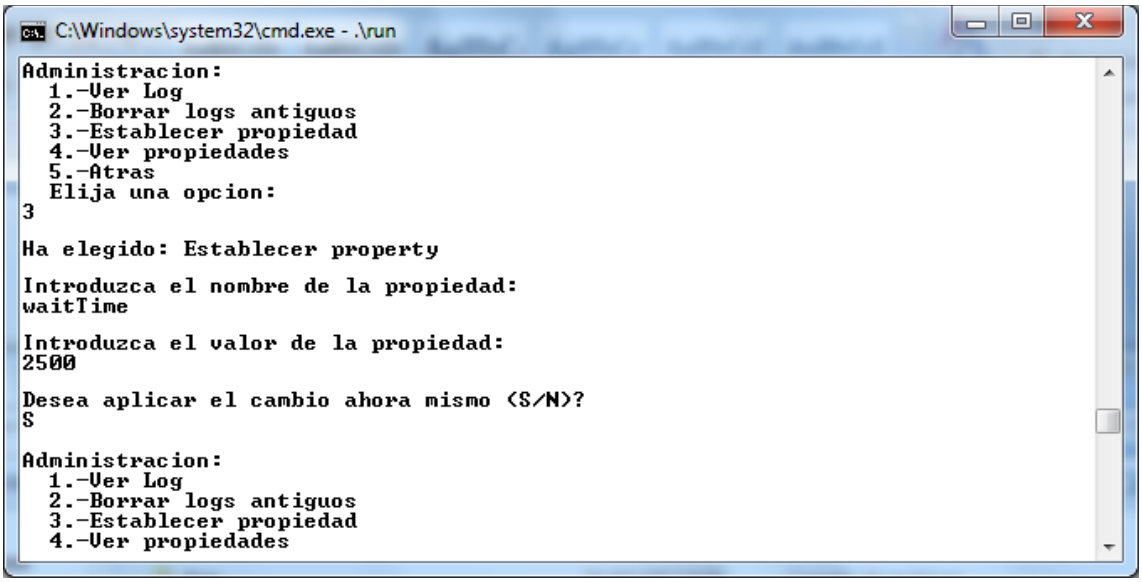


Figura 99: Modificación de una propiedad

Tal y como se puede ver en la figura 99, el usuario solo podrá modificar una propiedad cada vez que se ejecute esta opción del menú, y lo único que tendrá que conocer para realizar el cambio es el nombre exacto de la propiedad, que pueden ser: *ttl*, *waitTime*, *saveTime*, *pBest* o *userPath*.

El usuario también tendrá la opción de realizar el cambio de forma inmediata o de hacerlo efectivo la siguiente vez que se ejecute el sistema. Si se realiza de forma inmediata, el usuario podrá ver los cambios efectuados a través de la opción correspondiente del menú, que se pudo ver en el apartado anterior. El resultado sería el mostrado en la figura 100.

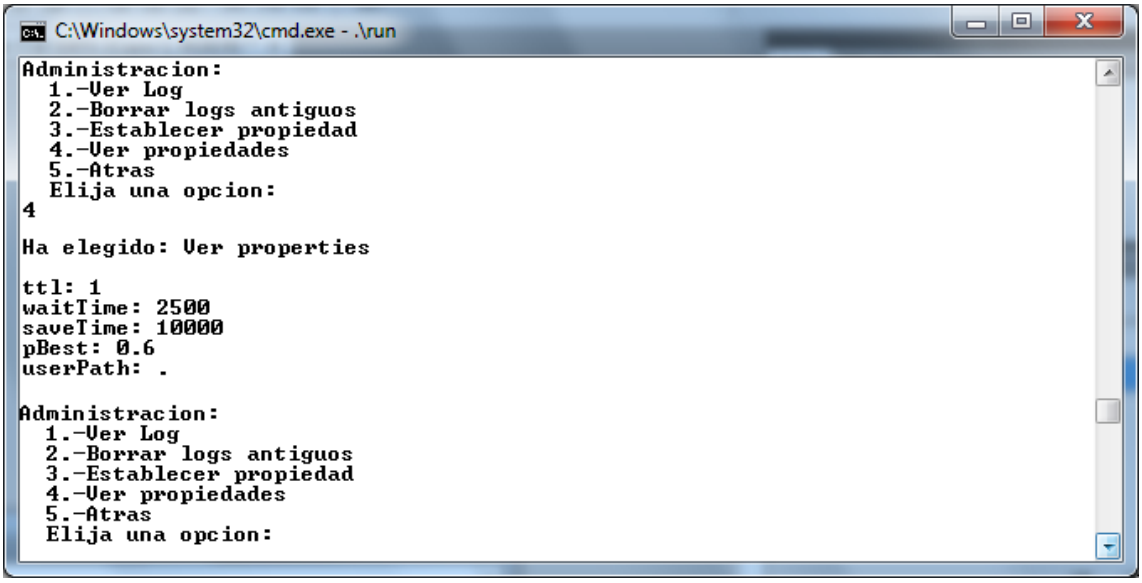


Figura 100: Resultado de la modificación de una propiedad

ANEXO D: MANUAL DEL DESARROLLADOR

En este anexo se complementa la información descrita en el capítulo 5, de forma que cualquier desarrollador pueda encontrar con todo detalle, la funcionalidad de cada uno de los atributos y métodos más importantes que conforman cada una de las clases del proyecto.

Al igual que ocurre con la implementación del sistema, este anexo estará estructurado en función de los módulos que lo forman, permitiendo conocer qué clases forman qué módulo y así facilitar la comprensión del conjunto.

D.1. Descripción de las interfaces

D.1.1. Interfaz de administrador

D.1.1.1. Métodos

- **`showLog`:**

  El sistema, para cada acción del usuario, bien sea con las tablas locales de confianza como con *peers* remotos, va guardando un registro con todos los cambios que se van produciendo en las tablas, y se van almacenando en un fichero de *log* que contendrá un registro de todos ellos.

  A lo largo de la ejecución, el usuario podrá acceder a las tareas de administración y consultar este fichero de *log*. Este método se encarga de devolver el contenido de dicho fichero en formato de cadena de texto para facilitar la presentación de la información al usuario.

- **`deleteOldLogFiles`:**

  La utilización de los ficheros de *log* no se centra exclusivamente en la creación y escritura en nuevos archivos sin control alguno. Hay que ser consciente de que el sistema puede estar ejecutándose en un dispositivo limitado que, por consiguiente, puede tener una capacidad de memoria ajustada. Por este motivo, o simplemente por querer llevar un registro ordenado sin información obsoleta, existe el método `deleteOldLogFiles`.

  Su misión es la de borrar todos los ficheros de *log* antiguos, cuya información estará desactualizada, quedando únicamente el fichero correspondiente a la ejecución actual del sistema.

- **`saveProperty`:**

  Como se explicó en su momento, mediante un fichero de configuración se podrán establecer algunos parámetros que determinarán ciertos comportamientos del sistema. Este fichero se consultará en el arranque del sistema, aunque podrá ser modificado en cualquier momento de la ejecución,

bien para que los cambios sean efectivos inmediatamente o bien para que sean tenidos en cuenta en la siguiente ejecución.

Para esto sirve precisamente el método `saveProperty`, se encarga de modificar el valor de una de las propiedades del sistema y de especificar si dicho cambio se deberá tener en cuenta en el acto.

- **`listProperties`:**

  En cualquier momento el usuario puede consultar los valores de dichas propiedades, ya sea por mera curiosidad o bien porque quiera comprobar que están configurados correctamente. Para ello, el método `listProperties` devolverá una cadena de caracteres con los valores asignados a todas las propiedades de configuración del sistema.

D.1.2. Interfaz de contenido

D.1.2.1. Métodos

- **`findContent`:**

  Este método le permite al usuario realizar la búsqueda de un contenido en el sistema. Dicho usuario solo tendrá que preocuparse de conocer la palabra clave en la que está interesado y el grupo de *peers* al que deseará encaminar la búsqueda. Con estos dos parámetros, el método tendrá suficiente información para comenzar su trabajo.

  Adicionalmente se le da la posibilidad al usuario de realizar directamente una búsqueda remota en la red. Esto se hará gracias a otro parámetro, un *booleano*, que indicará si se fuerza la búsqueda remota (verdadero) o si se quiere comenzar la búsqueda en las tablas locales (falso). Esto permitirá al usuario dirigir su búsqueda a contenidos nuevos presentes en la red.

  Los resultados de este método se organizarán en una tabla que contendrá, para cada grupo de intereses, una subtabla en la que cada entrada se corresponderá con las respuestas obtenidas de cada *peer* que recibió nuestra petición. Estas respuestas serán vectores con los identificadores de los contenidos que dichos *peers* poseen y que se ajustan a los parámetros de búsqueda.

- **`rateContent`:**

  Tras acceder a un contenido, el *peer* deberá realizar una evaluación del mismo de acuerdo con su criterio, puntuándolo con un valor numérico que podrá ir desde -1, en caso de que el contenido no haya cumplido ninguna de las expectativas del usuario, hasta 4, si el usuario ha visto en dicho contenido justo lo que estaba buscando.

  Para realizar esta evaluación se necesitará la palabra clave a la que hace referencia el contenido, el grupo al que se dirigió la búsqueda, el grupo en el que se encontró el contenido finalmente, el identificador del contenido, el

identificador del *peer* del que recibimos la respuesta y la puntuación que el usuario le da al contenido. En este punto se debe hacer hincapié en la presencia de dos identificadores de grupo: el primero de ellos está claro, ya que es aquel al cual se dirigió la búsqueda, pero el segundo es más complicado de entender. Este segundo grupo es aquel en el que el *peer* que contestó a la petición tenía asignado dicho contenido. Este, por lo general será el mismo que el primero, aunque habrá ocasiones en las que no habrá contenidos que se ajusten a la búsqueda en el grupo especificado, pero si en otros, por lo que se le podrán ofrecer de igual forma.

- **`addNewContent`:**

  El usuario del sistema será capaz de generar sus propios contenidos y compartirlos en red para el resto de los *peers*. En este caso, para introducir dicho contenido en el sistema de reputación deberemos utilizar este método, al que se le pasará el nombre del contenido, la palabra clave a la que hace referencia y el identificador del grupo en el que lo queremos dar a conocer. El resultado será el identificador con el que el sistema registra dicho contenido.

- **`addKeywordToContent`:**

  En ocasiones, un usuario que acaba de buscar un contenido y lo ha evaluado, puede darse cuenta de que dicho contenido hace referencia, además de a la palabra clave por la que lo buscó, a una segunda palabra clave que no había sido tenida en cuenta. En este caso, el usuario podrá invocar a este método para asignar a dicho contenido una nueva palabra clave. Esto hará que el contenido se publique en las tablas locales haciendo referencia a las dos palabras clave anteriores, y permitiendo que otros usuarios puedan encontrarlo buscando indistintamente por cualquiera de ellas. Los únicos parámetros que se necesitarán para ejecutar este método serán el identificador del contenido que se quiere etiquetar y la nueva palabra clave.

- **`removeContentFromGroup`:**

  Hay ocasiones en las que el usuario, que en su momento publicó un contenido en un determinado grupo de la red, quiere eliminarlo del mismo por diferentes motivos. Para este cometido está el método `removeContentFromGroup` el cuál, recibiendo como parámetro el identificador del grupo del que queremos eliminar el contenido, la palabra clave a la que hace referencia y su identificador, llevará a cabo el borrado. Como resultado, el método devolverá un valor *booleano* que indicará si la eliminación ha sido exitosa (verdadero) o si ha habido algún problema en el proceso (falso).

- **`getContentConfidenceValue`:**

  Hasta ahora se ha visto que un usuario puede buscar un contenido, crearlo, puntuarlo e incluso borrarlo, lo que hace bastante útil el sistema de cara a la gestión de contenidos. Sin embargo, en cualquier momento dicho usuario

puede estar interesado simplemente en conocer el valor de reputación correspondiente a un contenido en las tablas locales de confianza. Para recuperar este valor, el método `getContentConfidenceValue` necesitará conocer el identificador del contenido, el grupo en el que se desea hacer la consulta y la palabra clave a la que hace referencia. El resultado será el número de tipo `double` correspondiente a la reputación consultada.

D.1.3. Interfaz de peer

D.1.3.1. Métodos

- **`addPeersToTable`:**

  En ocasiones, bien sea porque un *peer* acaba de descubrir un grupo y necesita encontrar nodos pertenecientes al mismo para poder realizar consultas de contenidos, o bien porque un usuario quiera actualizar su tabla de confianza en *peers* para tener más probabilidades de encontrar contenidos en cada una de sus búsquedas, se hace necesaria la presencia de un método que realice esta actualización.

  El método `addPeersToTable` se encarga de encontrar nuevos *peers* en la red que pertenezcan a un grupo determinado, con el objetivo de añadirlos a sus tablas de confianza locales y, en futuras búsquedas, tener más fuentes capaces de proporcionar contenidos. Para ello lo único que necesita es el identificador *JXTA* del grupo que se desea actualizar. Internamente el sistema se encargará de buscar los métodos necesarios para actualizar las tablas mediante peticiones a *peers* remotos, bien sea a aquellos conocidos que se sepa que pertenecen a dicho grupo, o bien consultado en los anuncios recibidos *peers* que pertenezcan al mismo.

  Ya que el número de nuevas entradas de la tabla puede ser muy alto, y como es un método de simple actualización, en lugar de presentar los resultados obtenidos, el método se limitará a devolver un valor *booleano* que indicará si se han encontrado nueva información (verdadero) o si por el contrario no se han encontrado fuentes para actualizar la tabla (falso).

- **`getPeerConfidenceValue`:**

  Del mismo modo que en cualquier momento de la ejecución del sistema el usuario puede solicitar conocer el valor de reputación correspondiente a un contenido, se deberán aportar los métodos necesarios para que el mismo usuario sea capaz de consultar los valores locales de reputación de un *peer* determinado.

  Por este motivo surge la necesidad de `getPeerConfidenceValue`. Recordemos que para un mismo *peer* remoto podrán existir varios valores de reputación, dependiendo del ámbito en el que queramos realizar la consulta. Así, habrá un valor por cada grupo al que pertenezca dicho *peer* y por cada

palabra clave sobre la que se hayan intercambiado contenidos. Por este motivo, para el funcionamiento correcto de este método se necesita pasar como parámetro ambos datos, el identificador *JXTA* único del grupo en el que se quiere consultar y la palabra clave en formato cadena de caracteres.

Finalmente este método devolverá un valor numérico de tipo `double` correspondiente al valor de reputación almacenado localmente para dicho *peer* en el ámbito solicitado.

D.1.4. Interfaz de grupo

D.1.4.1. Métodos

- **`createNewGroup`:**

  En ocasiones, puede darse el caso de que un usuario, a pesar de pertenecer a varios grupos, tenga unos intereses específicos por algún tema no tratado en ninguno de ellos. Por este motivo tiene la opción de crear un nuevo grupo que se publicará en la red para permitir a otros usuarios que estén interesados en lo mismo publicar nuevos contenidos.

  Este método creará un nuevo grupo de intereses a partir de un nombre y una descripción, pasadas como cadenas de caracteres. El resultado del método será el identificador *JXTA* único que se haya asignado a dicho grupo.

- **`findNewGroup`:**

  Antes de recurrir al método anterior, el usuario puede realizar una búsqueda en la red para encontrar grupos a lo que unirse que ya estén creados por otros *peers*. Este método devolverá un vector con todos los identificadores *JXTA* de los grupos nuevos encontrados.

D.2. Módulo de gestión de tablas

D.2.1. ContentConfidence

D.2.1.1. Atributos

- **`keyword`:**

  Cada contenido versa sobre una temática diferente, e incluso puede ser que un mismo contenido trate varios temas. En cualquier caso, para hacer la diferenciación entre las calificaciones que se otorgan al mismo en función de cada palabra clave, se utiliza este atributo. Es una cadena de caracteres que permite agrupar los contenidos por intereses.

- **`contentName`:**

  Es el nombre del contenido, representado como una cadena de caracteres. A pesar de que la aplicación trabaja principalmente con los identificadores únicos asignados a cada contenido, se utiliza este atributo para que sea más amigable la presentación de la información al usuario.

- **`contentID`:**

  Este identificador es único para cada contenido, por lo que es el atributo que se utilizará en nuestro sistema para la gestión de valores de reputación, evitando así duplicidades y posibles confusiones en las búsquedas.

- **`peerGroupName`:**

  Es el nombre del grupo *JXTA* en el que se evalúa el contenido. Al igual que con los nombres de los contenidos, los nombres de grupo se utilizan para que la presentación de datos al usuario sea más amigable.

- **`peerGroupID`:**

  Al igual que ocurre en el caso de los contenidos, los grupos también están identificados unívocamente a través de identificadores de la plataforma *JXTA*. Esto puede resultar más complicado para presentar resultados al usuario, pero es más cómodo para la gestión interna del sistema.

- **`isLocal`:**

  Es un atributo de tipo *booleano* (verdadero o falso) que indica si el contenido ha sido creado localmente por el usuario. Este es un parámetro que permite al usuario llevar un control sobre los contenidos que ha ido creando a lo largo del tiempo y los que han sido obtenidos de otros nodos en la red.

- **`popularity`:**

  Es un número entero que indica el número de veces que el contenido ha sido accedido por lo que, tal y como su nombre indica, es una medida de la popularidad de dicho contenido. Es un parámetro que influye directamente en el cálculo de la reputación propiamente dicha del contenido.

- **`relevance`:**

  Es un número entero que representa el valor de reputación propiamente dicho del contenido. Es una cifra calculada a partir de la valor previo del mismo, y de los recibidos de otros nodos para el mismo contenido.

- **`timestamp`:**

  Es una marca de tiempo que indica la fecha y hora en la que se modificó por última vez el valor de reputación de dicho contenido. Este valor puede ser utilizado para hacer algún tipo de ponderación en los cálculos de actualización de los valores de reputación.

D.2.1.2. Métodos

- **`increasePopularity`:**

  Este método se invoca cada vez que se accede al contenido al que hace referencia, de forma que se aumente en uno el valor de popularidad del mismo.

- **`updateConfValueByRate`:**

  Realiza una actualización del valor de reputación del contenido. Este método se invoca cuando el usuario quiere actualizar el valor de reputación de un contenido al que acaba de acceder, referido a la misma palabra clave, y dentro del mismo grupo en el que el usuario local realizó la búsqueda.

- **`updateConfValueByFeedback`:**

  Al igual que el método anterior, actualiza el valor de reputación de un contenido. Este método es invocado cuando un *peer* remoto envía la nueva reputación asignada a un contenido que previamente recibió desde el sistema local. De esta forma, los valores de reputación de un contenido no serán estáticos aunque no se acceda a ellos de forma local, sino que siempre habrá una realimentación de todos los usuarios a los que se envíe dicho contenido. De esta forma, la actualización se hará en base a dos parámetros: el *feedback* del *peer* remoto, y la reputación que tengamos de dicho *peer*.

D.2.2. ContentConfidenceTable

D.2.2.1. Atributos

- **`contentConfTable`:**

  Es la tabla de confianza de los contenidos propiamente dicha. Tal y como se ha explicado antes, en esta tabla están recogidos todos los valores de reputación ordenados primero por grupo, esto es, grupo de intereses dentro de la red, y también organizado por palabra clave ya que, al fin y al cabo se realizarán búsquedas en el sistema por este campo. Como se puede ver en el diagrama, estos valores se almacenan como objetos de la clase `ContentConfidence`, ya vista en el anterior apartado.

- **`userPath`:**

  Es una cadena de caracteres que representa la ruta completa al directorio en el que el usuario almacena toda la información del sistema. Estará almacenada en un fichero de configuración, aunque podrá ser modificada en cualquier momento por el usuario.

- **`fileName`:**

  Es una cadena de caracteres que representa la ruta completa al fichero en el que se almacenará la tabla de `ContentConfidence`.

- **`userFile`:**

  Es la representación del fichero que contiene la tabla de `ContentConfidence`, facilitada por el lenguaje de programación Java. Es el objeto que se utilizará en última instancia para leer y escribir los valores almacenados en disco.

- **`contentIDTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los contenidos. En este caso particular se usa para buscar nombres de contenidos dado su identificador único.

- **`contentNameTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los contenidos. En este caso se utiliza para buscar los identificadores de los contenidos partiendo de su nombre, que conoceremos previamente.

- **`groupIDTable`:**

  Es una tabla auxiliar utilizada para facilitar la búsqueda de los grupos. Esta tabla permite buscar el nombre de un grupo dado su identificador en el sistema.

- **`groupNameTable`:**

  Es una tabla auxiliar que facilita la búsqueda de los grupos conocidos. Esta tabla permite buscar el identificador único de un grupo del sistema conociendo su nombre.

D.2.2.2. Métodos

- **`ContentConfidenceTable`:**

  Es el constructor de la clase. Recibe como parámetro la ruta en la que debe buscar la información almacenada en disco si la hubiese o, si es la primera vez que se utiliza el sistema, el directorio en el que se guardará la información periódicamente. Este constructor tiene por cometido inicializar el resto de atributos de la clase así como cargar en memoria todos los valores de reputación que pudiera haber almacenados.

- **`findContentConfidence`:**

  En este caso se debe tener cuidado ya que hay dos métodos con el mismo nombre. El primero de ellos recibe como parámetros un identificador de grupo y una plantilla de `ContentConfidence`, y sirve para comprobar que en dicho grupo existe tal valor de `ContentConfidence`.

  El segundo método con este nombre y recibe como parámetro una plantilla de `ContentConfidence`, esto es, un objeto de este tipo que tendrá definidos uno o más de los atributos de la clase. Estos atributos predefinidos serán los parámetros de búsqueda, de forma que se puede buscar por identificador de grupo, por identificador de contenido, por palabra clave, o por alguna combinación de estos tres. Si la búsqueda arroja algún resultado este se devolverá como un listado de `ContentConfidence`, cada uno de los cuales tendrá todos sus atributos cargados de acuerdo con la información almacenada localmente.

- **`getContentConfidence`:**

  Este es un método parecido al anterior, pero más específico. Se utiliza cuando tenemos claro el contenido que queremos consultar es decir, su identificador, el identificador del grupo en el que queremos buscar y la palabra clave a la que hace referencia, pero sólo nos falta el valor de reputación propiamente dicho. A diferencia del caso anterior, el método devuelve un objeto de tipo `ContentConfidence`, que contendrá el valor buscado.

- **`setContentConfidence`:**

  Este método se encarga de insertar en la tabla una `ContentConfidence` determinada. Sirve para labores de depuración. Recibe como parámetro un objeto de la clase `ContentConfidence` con todos sus atributos, de forma que se inserta tal cual en la tabla. El método devolverá un *booleano* que indicará si la operación se ha realizado con éxito.

- **`addContentConfidence`:**

  Este método permite insertar un nuevo `ContentConfidence` en la tabla. Recibe como parámetro el grupo en el que se debe insertar la nueva reputación, y devuelve un *booleano* que indicará si se ha agregado correctamente.

- **`updateContentConfidence`:**

  Al igual que ocurre con el método `findContentConfidence`, este está duplicado en nombre, aunque no en funcionalidad. El primero de estos métodos permite actualizar un valor de reputación en un contenido recibiendo como parámetro un objeto de tipo `ContentConfidence` del que obtendremos el identificador del contenido que queremos actualizar, el identificador de grupo en el que queremos realizar la actualización y la palabra clave en concreto a la que se hace referencia; también se pasará como parámetro un vector de `ContentConfidence` con los valores de reputación obtenidos de otros nodos para el mismo contenido, y por último una calificación subjetiva del usuario.

  Este método se invocará cuando el usuario haya buscado y accedido a un contenido, ya sea local u obtenido remotamente, y realice la calificación del mismo en base a su criterio personal, modificando el valor almacenado previamente para dicho contenido o creando uno nuevo en caso de no existir anteriormente.

  El segundo de los métodos que permiten actualizar un valor de reputación en contenido. En este caso el método se invocará cuando el sistema reciba el *feedback* de un contenido que habremos facilitado previamente a un *peer* remoto de la red; dicho *peer* accederá al contenido y lo calificará tal y como

se haría al buscar un contenido local, enviando la nueva calificación actualizada que almacenará en su tabla local.

`updateContentConfidence`, por lo tanto, recibe como parámetros una plantilla que indicará el contenido al que hace referencia la reputación, el *feedback* recibido del *peer* remoto y la reputación que tiene el sistema para el *peer* remoto, para ponderar dicho *feedback*.

- **`getcontentConfidenceValue`:**

  Este método permite acceder directamente al valor numérico entero que corresponde a la reputación que está almacenada en la tabla local para un contenido determinado. Recibe como parámetro el identificador del grupo en el que queremos hacer la búsqueda y una plantilla de `ContentConfidence` que contendrá el identificador del contenido que queremos buscar y la palabra clave a la que hace referencia, así como el identificador de grupo en el que se desea hacer la consulta.

- **`addGroupTable`:**

  La utilidad de este método es la de actualizar los valores de reputación de todos los contenidos dentro de un grupo dado y para cualquier palabra clave, insertando en la tabla un nuevo conjunto de entradas y, posiblemente escribiendo sobre los valores anteriores.

  La invocación de este método no se suele realizar directamente para grupos en los que ya tenemos valores de reputación locales, ya que esto implicaría una pérdida de toda la información obtenida mediante las calificaciones propias del usuario, sino que se le llamará cuando se quiera recabar la información de contenidos para un grupo nuevo recibida de un *peer* remoto.

- **`getKeywordsForAdv`:**

  Este método, aunque no parezca ser uno de los más relevantes, sí que tiene vital importancia en la publicación en la red de la información local de cara a que nodos remotos puedan acceder a la información que se almacena en las tablas locales. `getKeywordsForAdv` se encarga de recopilar toda la información de todos los contenidos almacenados localmente y crear un documento *XML* que contendrá, para cada grupo al que pertenezca el usuario, todas las palabras clave sobre las que tenga algún contenido.

- **`getAvailableGroups`:**

  Al igual que el anterior, este método es especialmente importante para el funcionamiento del sistema, ya que permite obtener un listado con todos los grupos a los que pertenece el usuario, mediante un recorrido por todas las tablas de contenido locales. Es un método que facilita la presentación de la información de los grupos para que la búsqueda de contenidos sea más cómoda.

- **`removeContent`:**

  Como se puede deducir del nombre, se encarga del borrado de un contenido almacenado localmente. Para este cometido, el método debe recibir como parámetro el identificador del grupo del que se desea borrar el contenido, la palabra clave a la que hace referencia y el identificador del contenido propiamente dicho. Para conocer el resultado de la operación devolverá un valor *booleano* que indicará si el contenido se ha borrado correctamente.

- **`getContentName`:**

  Este método auxiliar permite obtener el nombre de un contenido almacenado localmente a partir de su identificador único.

- **`getContentID`:**

  Este método permite obtener el identificador *JXTA* de un contenido almacenado localmente pasando como parámetro el nombre del mismo.

- **`getGroupname`:**

  Devuelve el nombre de un grupo dado su identificador único.

- **`getGroupID`:**

  Permite obtener el identificador único de un grupo pasando como parámetro su nombre.

- **`addContentPair`:**

  Añade a las tablas auxiliares `contentNameTable` y `contentIDTable` un nuevo par nombre-identificador de contenido.

- **`addPeerGroupPair`:**

  Añade a las tablas auxiliares `groupNameTable` y `groupIDTable` un nuevo par nombre-identificador de grupo.

- **`toXMLFile`:**

  Este método, junto con los de búsqueda y almacenamiento de la información en las tablas, es uno de los más importantes de la clase, ya que permite el guardado en disco de toda la información local. Este guardado se realiza periódicamente para evitar pérdidas de información causadas por hipotéticas caídas del sistema, permitiendo tener siempre un contenido suficientemente actualizado en disco.

  Del mismo modo, este método será invocado cuando el usuario decida finalizar su sesión en el sistema.

- **`fromXMLFile`:**

  Se invoca cada vez que el sistema arranca, y su cometido es cargar en memoria las tablas de confianza en contenidos que están almacenados en disco, de forma que su acceso y actualización sean más rápidos.

D.2.3. PeerConfidence

D.2.3.1. Atributos

- **`keyword`:**

  Como ocurría con `ContentConfidence`, el valor de reputación de un *peer* tendrá sentido únicamente si va a acompañado por la palabra clave a la que hace referencia, de forma que pueden existir, para un mismo *peer*, diferentes valores de reputación en función de la palabra. Esto quiere decir, en resumen, que un determinado *peer* puede ser muy bueno en cuestiones de economía, pero no serlo tanto en temas de deporte, y por lo tanto se debe realizar una distinción en función de lo que se esté buscando.

- **`peerID`:**

  Es el identificador único *JXTA* que nos permite determinar la identidad del *peer* en concreto al que hace referencia la reputación que estamos tratando.

- **`peerName`:**

  Como ocurre en otros casos, este es el nombre, en formato cadena de texto, correspondiente al *peer*. De nuevo, este atributo se utiliza para facilitar la presentación de datos al usuario.

- **`peerGroupID`:**

  El valor de reputación del *peer* tiene que ir acompañado del identificador del grupo en el que dicho valor tiene sentido. Este atributo es el identificador *JXTA* único para el grupo al que hace referencia el valor de reputación del *peer* que estamos considerando.

- **`peerGroupName`:**

  Es una cadena de texto que representa el nombre del grupo.

- **`confidenceValue`:**

  Es un número entero que representa el valor de la reputación del *peer* propiamente dicho.

- **`popularity`:**

  Es un número entero que indica el número de contenidos que el *peer* al que hace referencia el valor de reputación nos ha enviado, y que hemos calificado. Como su nombre indica es una medida de la popularidad subjetiva que tiene dicho *peer* en nuestro sistema.

- **`timeStamp`:**

  Es una marca de tiempo que indica el instante en que el valor de reputación para el *peer* se creó o fue modificado por última vez. Esto nos da una idea de lo reciente que es dicho valor, y por lo tanto, puede verse como una medida de la calidad de dicho valor.

- **`sumContentConf`:**

  Es un número de tipo `double` que representa la suma de las calificaciones de todos los contenidos obtenidos de este *peer*. Este valor nos ayudará a calcular el valor de reputación del *peer*.

- **`numContentConf`:**

  Es un número entero que representa el número de contenidos obtenidos del *peer* al que hace referencia este valor de reputación en total. Este valor, al igual que el anterior será de utilidad para realizar los cálculos de actualización de la reputación.

D.2.3.2. Métodos

- **`updateConfValue`:**

  Este método se encarga de actualizar la reputación que se tiene del nodo al que hace referencia el objeto mediante la reputación del último contenido que nos ha enviado el mismo. De esta forma, la reputación de un *peer* depende única y exclusivamente de la calidad de los contenidos que nos ha proporcionado a lo largo del tiempo, y no de calificaciones directas que pudieran dar lugar a valores de reputación adulterados.

  De esta forma, si un *peer* nos envía contenidos de calidad que, como es lógico, calificaremos con buenas puntuaciones, la reputación de dicho nodo aumentará, mientras que si los contenidos recibidos de este *peer* son de mala calidad o no se ajustan a lo que dicen ser, la reputación de este nodo irá cayendo y se terminará por prescindir del mismo como fuente de contenidos.

D.2.4. PeerConfidenceTable

D.2.4.1. Atributos

- **`peerConfTable`:**

  Es la tabla, propiamente dicha, que almacena los valores de reputación en los *peers* con los que el sistema ha ido teniendo algún intercambio de información a lo largo del tiempo. Como se ha mencionado antes, está basada en el uso de una `Hashtable`, clase proporcionada por el *API* de *Java*, que nos permitirá organizar la tabla por grupos de intereses, que recordemos son equivalentes a los grupos *JXTA*, y en un segundo nivel por palabras clave, facilitando de este modo su búsqueda.

- **`userPath`:**

  Es una cadena de caracteres que representa la ruta completa al directorio en el que el usuario almacena toda la información del sistema. Estará almacenada en un fichero de configuración, aunque podrá ser modificada en cualquier momento por el usuario. Como se vió en los requisitos previos del sistema, este directorio deberá contar con permisos de lectura y escritura para el usuario.

- **`fileName`:**

  Es una cadena de caracteres que representa la ruta completa al fichero en el que se almacenará la tabla de `PeerConfidence`.

- **`userFile`:**

  Es la representación del fichero que contiene la tabla de `PeerConfidence`, facilitada por el lenguaje de programación *Java*. Es el objeto que se utilizará en última instancia para leer y escribir los valores almacenados en disco.

- **`peerIDTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los *peers*. En este caso particular se usa para buscar el nombre de un *peer* dado su identificador *JXTA* único.

- **`peerNameTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los *peers*. En este caso se utiliza para buscar los identificadores de los *peers* pasándole como parámetro su nombre, que conoceremos previamente.

- **`groupIDTable`:**

  Es una tabla auxiliar utilizada para facilitar la búsqueda de los grupos. Esta tabla permite buscar el nombre de un grupo dado su identificador en el sistema.

- **`groupNameTable`:**

  Es una tabla auxiliar que facilita la búsqueda de los grupos conocidos. Esta tabla permite buscar el identificador único de un grupo del sistema conociendo su nombre.

D.2.4.2. Métodos

- **`setUserPath`:**

  Es un método trivial que se encarga de actualizar la cadena de caracteres que representa la ruta del directorio de usuario, y por lo tanto donde se escribirá la información de la tabla.

- **`updatePeerConfidence`:**

  Este método, como su nombre indica, se encarga de la actualización de los valores de reputación de *peer* que están almacenados en memoria. Para hacerlo necesita dos parámetros: el primero de ellos será una plantilla, en forma de objeto de tipo `PeerConfidence`, que indicará el *peer* del que deseamos actualizar su reputación, y el grupo y la palabra clave para la que queremos hacerlo; el segundo de estos parámetros es el valor de reputación asignado al último contenido que nos proporcionó dicho *peer*.

  Como se ha comentado, no hace falta nada más que esto para realizar la actualización, ya que se obtendrá del valor de reputación del último contenido toda la información necesaria para este cometido, tal y como se explicará en el apartado de desarrollos algorítmicos.

  El método `updatePeerConfidence` se invocará cada vez que se realice la búsqueda de un contenido, inmediatamente después de la actualización de su valor de reputación, para actualizar de igual forma la reputación del *peer* que nos lo envió. Como se puede observar esto hace que los valores de reputación de los *peers* sean valores muy cambiantes en el tiempo, ya que continuamente se estarán realizando búsquedas y calificaciones de contenidos.

- **`findPeerConfidence`:**

  La utilidad de este método también queda delatada por su nombre y es, precisamente, la de realizar una búsqueda de una `PeerConfidence` en las tablas locales. Para hacerlo, al igual que el método anterior, recibe como parámetro una plantilla de tipo `PeerConfidence` en la que estarán definidos los parámetros de búsqueda, pudiendo ser: identificador del *peer*, identificador de grupo o palabra clave.

  La búsqueda puede realizarse para todos ellos, lo que implicaría el más restrictivo de los casos, o un subconjunto de uno o dos de ellos, caso más frecuente en la ejecución del sistema. Por tanto, `findPeerConfidence` devolverá un vector con todos los resultados que se ajusten a la búsqueda. En el primero de los casos se puede obtener un vector con uno o ningún elemento, mientras que en el segundo caso el abanico de resultados puede llegar a ser mucho mayor.

  Este método se invocará cada vez que el *peer* acceda a un contenido nuevo. Esto nos permitirá saber a qué *peers* preguntar por contenidos dentro de un grupo, y que tengan relación con una palabra clave determinada. Esto tendrá una doble utilidad, ya que, en primer término se tendrá un registro de todos los identificadores de los *peers* conocidos, y por otro lado también estarán almacenados todos sus valores de reputación para ordenarlos según los resultados de las interacciones pasadas con ellos.

- **`getPeerConfidence`:**

  Este método es muy similar al anterior, pero realiza una búsqueda completamente restrictiva en la que se definirá el identificador del *peer* que queremos buscar, el identificador del grupo donde queremos realizar la búsqueda y la palabra clave a la que se refiere, todo ello definido en un objeto de la clase `PeerConfidence`. La diferencia en este caso estriba en que en lugar de un vector de resultados, se espera una única coincidencia.

  Este método se suele invocar cuando se desea conocer la reputación de un *peer* en concreto en unas determinadas circunstancias. Generalmente suele ser para presentar dicha información detallada al usuario del sistema, cuando este desea realizar una consulta específica.

- **`setPeerConfidence`:**

  Este método se encarga de establecer el valor de la reputación de un *peer* directamente en las tablas de memoria. Recibirá como parámetro la `PeerConfidence` que se desee insertar en la tabla, devolviendo un valor *booleano* que indicará si la operación ha sido exitosa.

  Este es un método podría calificarse de depuración, ya que nunca se va a querer imponer un valor de reputación directamente, sino que siempre se realizará vía actualización con el método `updatPeerConfidence` ya conocido, actualizando el valor que existiese previamente o creando uno nuevo en caso de ser necesario.

- **`getPeerName`:**

  Devuelve el nombre de un *peer* dado basándose en el identificador *JXTA* del mismo, que recibe como parámetro. La búsqueda se realizará sobre la tabla auxiliar `peerIDTable`.

- **`getPeerID`:**

  Devuelve el identificador *JXTA* único de un *peer* realizando una búsqueda sobre la tabla auxiliar `peerNameTable`. Este método recibe como parámetro una cadena de texto que representa el nombre del *peer*.

- **`getGroupName`:**

  Devuelve el nombre de un grupo en formato cadena de caracteres dado su identificador *JXTA*. La búsqueda se realiza sobre la tabla auxiliar `groupIDTable`.

- **`getGroupID`:**

  Permite obtener el identificador único *JXTA* de un grupo pasando como parámetro su nombre. Esta búsqueda se realiza sobre la tabla auxiliar `groupNameTable`.

- **`addPeerPair`:**

  Añade a las tablas auxiliares `peerNameTable` y `peerIDTable` un nuevo par nombre-identificador de *peer*.

- **`addPeerGroupPair`:**

  Añade a las tablas auxiliares `groupNameTable` y `groupIDTable` un nuevo par nombre-identificador de grupo.

- **`toXMLFile`:**

  Este método, junto con los de búsqueda y almacenamiento de la información en las tablas, es uno de los más importantes de la clase, ya que permite el guardado en disco de toda la información local. Este guardado se realiza periódicamente para evitar pérdidas de información causadas por hipotéticas caídas del sistema, permitiendo tener siempre un contenido suficientemente actualizado en disco.

  Del mismo modo, este método será invocado cuando el usuario decida finalizar su sesión en el sistema.

- **`fromXMLFile`:**

  Se invoca cada vez que el sistema arranca, y su cometido es cargar en memoria las tablas de confianza de *peer* que están almacenados en disco, de forma que su acceso y actualización sean más rápidos.

D.2.5. PeerConfidenceTableGI

D.2.5.1. Atributos

- **`peerConfTableGI`:**

  Es la tabla de confianza en *peers* independiente de grupos propiamente dicha. De nuevo en este caso se ha optado por basarse en el uso de objetos de la clase `Hashtable` proporcionados por el *API* de *Java*, que nos permitirá realizar una organización de los datos según cadenas de caracteres que se corresponderán con las distintas palabras clave a las que hacen referencia los contenidos para facilitar su búsqueda. En un segundo nivel, los datos de este atributo se organizan según el identificador *JXTA* del *peer* al que hace referencia, evitando de este modo tener que recorrer todos los datos hasta obtener el buscado.

- **`userPath`:**

  Este es un atributo ya visto en las diferentes tablas de información repasadas hasta ahora, y que tiene idéntica función: la de representar el directorio en el que el usuario almacenará la información.

- **`fileName`:**

  Al igual que el anterior, ya conocemos el cometido de esta cadena de caracteres, es la representación de la ruta completa al fichero que contendrá la tabla de confianza en *peer* independiente de grupo.

- **`userFile`:**

  Es la representación *Java* del fichero propiamente dicho y será el que permitirá leer y escribir en disco toda la información almacenada en memoria.

- **`peerIDTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los *peers*. En este caso particular se usa para buscar el nombre de un *peer* dado su identificador *JXTA* único.

- **`peerNameTable`:**

  Es una tabla auxiliar utilizada para facilitar las búsquedas de los *peers*. En este caso se utiliza para buscar los identificadores de los *peers* pasándole como parámetro su nombre, que se conocerá previamente.

D.2.5.2. Métodos

- **`setUserPath`:**

  Este es un método exactamente igual que los de las clases previas `PeerConfidenceTable` y `ContentConfidenceTable`, permitiendo establecer la ruta al directorio de trabajo del usuario.

- **`findPeerConfidence`:**

  Como en los casos anteriores este es uno de los métodos más importantes de la clase, debido a que es el que lleva toda la carga de la recuperación de valores de reputación en *peers* de la tabla. Al igual que sus homólogos en otras clases, este método recibe como parámetro la plantilla de tipo `PeerConfidence` en la que se podrán especificar los parámetros para realizar la búsqueda que, en este caso, serán exclusivamente identificador del peer y palabra clave (recordemos que en este caso no tenemos en cuenta los grupos).

  Al igual que ocurría con los otros métodos, este devolverá un vector con todas las coincidencias encontradas en la tabla, pudiendo ser un vector vacío, con uno, o con varios resultados.

- **`getPeerConfidence`:**

  Este es un caso particular del método anterior, en el que los parámetros de búsqueda: palabra clave e identificador de *peer*, estarán definidos siendo pasados como parámetro. El resultado en este caso será un único resultado de tipo `PeerConfidence`.

- **`setPeerconfidence`:**

  La función de este método es la de establecer un valor `PeerConfidence` dentro de la tabla. Como ocurría con otras clases este es un método de depuración ya que toda modificación de los valores almacenados debería hacerse vía el método `updatePeerConfidence`, que veremos a continuación.

- **`updatePeerConfidence`:**

  Este método realiza la misma operación de actualización del valor numérico decimal de una `PeerConfidence` almacenada en la tabla, mediante el algoritmo correspondiente. Recibe como parámetros una plantilla `PeerConfidence` que nos indicará el objeto que se tiene que actualizar y un `ContentConfidence` que se corresponde con el valor de reputación del último contenido intercambiado con el *peer* al que hace referencia la `PeerConfidence`.

- **`getPeerName`:**

  Este es un método auxiliar que se encarga de buscar en la tabla `peerIDTable` el nombre de un *peer* determinado. Se le pasa como parámetro el identificador único de un *peer* y devuelve como parámetro la cadena de caracteres que representa el nombre del mismo.

- **`getPeerID`:**

  Tomando como referencia la tabla auxiliar `peerNameTable`, este método se encarga de devolver el identificador *JXTA* único de un *peer* determinado pasándole como parámetro su nombre en formato cadena de caracteres.

- **`addPeerPair`:**

  Para complementar a los dos métodos anteriores existe `addPeerPair`, que se encargará de ir almacenando en las tablas `peerIDTable` y `peerNameTable` las entradas Identificador-Nombre de cada *peer* con el que el sistema haya tenido algún contacto.

- **`toXMLFile`:**

  De nuevo nos encontramos con el método encargado de guardar en disco toda la información contenida en la tabla que maneja la clase. El guardado será periódico para evitar posibles pérdidas importantes de información y también se ejecutará a la hora de cerrar la sesión.

- **`fromXMLFile`:**

  Este método se invocará cada vez que el usuario entre en el sistema y su cometido será el de cargar en memoria la tabla de confianza en *peers*

independiente de grupo, inicializando el atributo `peerConfTableGI` con los valores almacenados en disco y conocidos hasta el momento.

D.2.6. Risk

D.2.6.1. Atributos

- **`peerID`:**

  Es el identificador *JXTA* del *peer* al que hace referencia el riesgo. Como se puede ver en el diagrama *UML* de la figura 110, es una instancia de la clase `PeerID` contenida en el *API* proporcionado por la plataforma *JXTA*.

- **`peerName`:**

  Es una cadena de caracteres que representa el nombre del *peer*. Ya hemos visto que este atributo está presente en la mayoría de las clases básicas anteriores, dada su utilidad dentro del sistema para la presentación de información al usuario. En este caso también será un objeto de tipo `String`.

- **`integrity`:**

  Es uno de los componentes básicos de la clase. Es un número decimal en el rango [-1, 4] que representa el estado de los contenidos recibidos. Por ejemplo, si observamos que todos los contenidos que nos ha enviado un *peer* contienen algún tipo de virus, consideraremos que todos ellos están en mal estado, y por lo tanto este componente tendrá un valor muy cercano a -1.

- **`accesibility`:**

  Es el segundo de los tres componentes básicos de la clase. Es un número entero decimal entre -1 y 4 que representa el grado de accesibilidad que tiene el *peer*, es decir, es una medida de las veces que se ha intentado acceder a dicho *peer* y, efectivamente, hemos recibido su respuesta.

- **`performance`:**

  Es el último de los atributos. Al igual que los dos anteriores es un valor numérico decimal comprendido entre -1 y 4, pero en este caso representa una medida del rendimiento del *peer*, expresada como el tiempo de respuesta del mismo. A mayor velocidad, mejor rendimiento, y por lo tanto esta puntuación se aproximará más a 4 cuanta mayor rapidez demuestre el *peer*.

D.2.6.2. Métodos

- **`updateAccesibility`:**

  Es el método encargado de actualizar el atributo `accesibility` correspondiente a dicho *peer*. Recibe como parámetro un número decimal de tipo `double` que representará si la última comunicación con dicho *peer* fue satisfactoria, es decir, si se consiguió contactar con el mismo.

- **`updatePerformance`:**

  Este método actualiza la calificación del rendimiento del *peer*. Se calculará a partir de un número decimal pasado como parámetro que representará la calidad de la velocidad de respuesta del *peer* remoto, y se combinará con el valor almacenado previamente para obtener un nuevo valor de rendimiento.

D.2.7. RiskTable

D.2.7.1. Atributos

- **`riskTable`:**

  Es la tabla de riesgos propiamente dicha, representada como una `Hashtable` (clase proporcionada por el *API* de *Java*), que se organiza tomando como clave principal el identificador del *peer* al que hace referencia cada elemento.

- **`userPath`:**

  Como ocurre con todas las tablas, se necesita saber el directorio de trabajo donde se guardará toda la información del usuario, y por lo tanto el atributo `userPath`, que representa la ruta a este directorio también estará presente en la clase. Al igual que en el resto de ocasiones este atributo será una cadena de caracteres (objeto `String` de *Java*).

- **`fileName`:**

  Este atributo también es común a todas las clases que modelan las tablas, representando la ruta completa al fichero en el que se almacenará la tabla de objetos `Risk`. De nuevo será una cadena de caracteres.

- **`userFile`:**

  Este último atributo es igual que los dos anteriores, es decir, común a todas las tablas, siendo un objeto *Java* de la clase `File`, representación del fichero en el que se almacenará la tabla.

D.2.7.2. Métodos

- **`RiskTable`:**

  Es el constructor de la clase y se encarga de la inicialización de los atributos de la misma. Recibe como parámetro la cadena de caracteres que representa al directorio de usuario, a partir de la cual se obtendrá la ruta del fichero y el propio fichero. Este constructor se encarga de inicializar el objeto `riskTable`, pero no de cargar los datos en él.

- **`setUserPath`:**

  El usuario del sistema puede elegir cambiar el directorio de trabajo en cualquier momento, y este método es el encargado de actualizar la ruta a

dicho directorio, recibiendo como parámetro su nuevo valor. Este método también se encargará de modificar los valores de `fileName` y `userFile`.

- **`getRisk`:**

  Es el encargado de obtener un objeto de la clase `Risk` concreto. Para ello, y de forma similar al resto de tablas del sistema, el método recibe como parámetro una plantilla en forma de objeto `Risk` que tendrá únicamente el identificador del *peer* cuyo factor de riesgo queremos consultar. Este método devolverá otro objeto de la clase `Risk` con todos los atributos cargados.

- **`setRisk`:**

  Al igual que ocurre con todas las tablas, este es un método de depuración que permite establecer directamente el valor de una entrada de la tabla, con unos atributos determinados. Recibe como parámetro un objeto de la clase `Risk` completo, es decir, con todos los atributos con valores concretos.

- **`findRisk`:**

  De nuevo en esta clase nos volvemos a encontrar con un método duplicado en nombre aunque, como suele ocurrir, no en funcionalidad. Este primer método recibe como parámetro una plantilla de `Risk`, y se encarga de buscar las entradas de la tabla que concuerdan con el patrón y devolverlas dentro de un vector de resultados. Este método se puede invocar cada vez que el sistema intente acceder a un nuevo contenido, con el objeto de elegir al mejor *peer* que nos lo facilite.

  El segundo método con este nombre es similar al anterior, recibiendo como parámetro una plantilla exactamente igual, pero en este caso devolviendo un valor numérico decimal que representará el factor de riesgo que conlleva interactuar con el *peer* que buscamos. Este valor se calcula como una media aritmética de los valores de `integrity`, `accesibility` y `performance`. Este valor será modificado para entrar dentro del rango [0,4] mediante la transformación matemática vista en el apartado de desarrollos algorítmicos. Este método se puede invocar cuando se desee elegir al mejor de los *peers* que nos puedan ofrecer un mismo contenido.

- **`updateRisk`:**

  Este método se invocará cada vez que se haya accedido a un contenido y se haya calificado. Este proceso, como hemos visto, además de provocar la actualización del `ContentConfidence` y el `PeerConfidence` correspondientes conllevará la actualización del factor de riesgo de un *peer*. Para esto, el método necesita como parámetros el identificador y el nombre del *peer* al que hace referencia el riesgo, y las medidas de accesibilidad y rendimiento que ha tenido el *peer* en su última actuación. Devuelve como resultado un *booleano* que indicará si la actualización ha sido correcta o no.

- **`toXMLFile`:**

  Este método es también común a todas las clases que representan tablas, y es el que permite almacenar toda la información del riesgo como un fichero *XML* situado en el directorio de trabajo del usuario. De nuevo en este caso, se invocará periódicamente para evitar pérdidas de información y al término de la sesión de usuario para guardar los últimos datos disponibles.

- **`fromXMLFile`:**

  Igual que el anterior, este método se puede encontrar en todas las clases correspondientes a las tablas del sistema. Se invocará al iniciar el sistema, provocando que la información almacenada en disco sobre los riesgos sea cargada en memoria.

D.2.8. TableManager

D.2.8.1. Atributos

- **`ccTable`:**

  Es un objeto de la clase `ContentConfidenceTable` que, como vimos anteriormente, representa la tabla de confianza en contenidos que manejará el sistema a lo largo de su vida, y que será actualizada con cada nueva observación del usuario.

- **`pcTable`:**

  Es un objeto de la clase `PeerConfidenceTable` que representa la tabla de confianza en *peers* para el usuario que ejecuta el sistema, de forma que recoge toda la información subjetiva que ha derivado de todas las acciones pasadas del mismo.

- **`riskTable`:**

  Es un objeto de la clase `RiskTable`, es decir, que representa la tabla de factores de riesgo con las observaciones personales del usuario del sistema.

- **`pcTableGI`:**

  Como vimos anteriormente, las tablas de confianza en *peers* pueden ser independientes de grupo, en concreto, este es un objeto de la clase `PeerConfidenceTableGI` que recogerá todas la información de reputación de *peers* que ha obtenido el sistema a lo largo de su vida, pero resumida de forma independiente a los grupos a los que pertenece.

D.2.8.2. Métodos

- **`TableManager`:**

  Es el constructor de la clase. Se encarga de la inicialización de las tablas pasándoles como parámetro la ruta completa al directorio de trabajo del

usuario que, a su vez, también recibirá como parámetro. También se encarga de invocar a los métodos `fromXMLFile` de todas ellas para cargar toda la información almacenada en disco, dejando así el sistema listo para su uso.

- **`setUserPath`:**

  Como ya se ha visto, el usuario tiene la posibilidad de cambiar su directorio de trabajo en cualquier momento de la ejecución del sistema. Para poder transmitir este cambio a cada una de las tablas que hay por debajo, este método se encarga de invocar los métodos del mismo nombre en cada uno de los atributos, haciendo efectivo este cambio.

- **`getKeywordDocument`:**

  Este método nos resulta familiar, ya que lo vimos contenido en la clase `ContentConfidenceTable`, de hecho, invoca a dicho método del atributo `ccTable` y devuelve los resultados en formato de documento *XML* con toda la información de las palabras clave a las que hacen referencia los contenidos almacenados localmente.

- **`getAvailableGroups`:**

  Al igual que el método anterior, este invoca a su homónimo del atributo `ccTable` para obtener una enumeración, como un objeto del tipo `Enumeration` del *API* de *Java*, de todos los identificadores de los grupos sobre los que tiene información dicho *peer*, presumiblemente son todos los grupos a los que el *peer* pertenece.

- **`updateContentConfidence`:**

  Al igual que ocurre en otras clases, este es otro ejemplo de un método *sobrecargado*, es decir, existen varios métodos en la clase con el mismo nombre pero que reciben distintos parámetros. En este primer método, los parámetros que se reciben son la palabra clave a la que hace referencia la `ContentConfidence` a actualizar, el nombre y el identificador *JXTA* del contenido, el nombre y el identificador único del grupo en el que se desea hacer la actualización, un vector con todas las `ContentConfidence` recibidas de otros *peers* para el mismo caso que queremos actualizar, el identificador del *peer* que nos proporcionó dicho contenido y el número decimal de tipo `double` correspondiente a la calificación que el usuario le ha asignado a dicho contenido.

  Este `updateContentConfidence` se encargará de crear una plantilla de tipo `ContentConfidence` para pasárselo al método correspondiente del atributo `ccTable`, devolviendo un *booleano* que indicará si la actualización ha sido correcta o no.

Este método en concreto se invocará cuando el usuario haya buscado y accedido a un contenido, ya sea remoto o local, y desee actualizar el antiguo valor de reputación del mismo.

El segundo método con este mismo nombre recibe como parámetros, en su mayoría, los mismos que en el caso anterior, pero con alguna diferencia: la palabra clave para la que se quiere puntuar el contenido, el identificador único y el nombre del contenido, el identificador *JXTA* y el nombre del grupo en el que se quiere evaluar el contenido, un objeto de tipo `ContentConfidence` con la opinión de otro *peer* sobre el mismo contenido, y el nombre y el identificador *JXTA* de dicho *peer*.

Este método, a diferencia del anterior, se invoca únicamente cuando nuestro sistema ha proporcionado un contenido a un *peer* remoto. Dicho *peer* habrá calificado el contenido y modificado sus tablas de confianza para dicho contenido y para el usuario de nuestro sistema, y después nos habrá enviado la nueva `ContentConfidence` que ha calculado para poder actualizar las tablas locales en consecuencia.

En este caso también se devuelve como resultado un valor *booleano* que indicará si la actualización del contenido ha sido exitosa.

- **`findContentByKeyword`:**

  Este método constituye la base de las búsquedas de contenidos dentro de las tablas locales. Cuando un usuario quiere encontrar un contenido en el sistema, lo que debe hacer es invocar a este método pasándole como parámetro la palabra clave a la que hace referencia el contenido que se está buscando, y el identificador *JXTA* único del grupo en el que se desea hacer la búsqueda. En base a esto, `findContentByKeyword` realiza una búsqueda en la tabla local de confianza en contenidos, representada en este caso por el atributo `ccTable`, y devuelve un vector con todos los `ContentConfidence` que se ajustan a las especificaciones de la petición.

  Como se ha mencionado, este método se invocará siempre que el usuario quiera hacer una nueva búsqueda de contenidos, siempre y cuando no fuerce una búsqueda en las tablas de los usuarios remotos, en cuyo caso la búsqueda será distinta.

- **`findPeerByKeyword`:**

  Este método también se invoca cuando el usuario realiza una nueva búsqueda de contenidos, pero solo en dos casos concretos: el primero de ellos se da cuando el usuario, tras buscar en sus tablas locales de confianza en contenidos, no encuentra ningún resultado que se ajuste a su búsqueda, y por lo tanto debe continuar intentándolo con los *peers* de la red; el segundo de ellos es cuando el usuario quiere forzar una búsqueda remota, lanzando directamente la petición a los *peers* que conoce en la red.

Los parámetros que recibe son los mismos que el método anterior, es decir, la palabra clave a la que hace referencia el contenido que se busca, y el grupo de intereses en el que se quiere realizar la búsqueda. Sin embargo, en este caso se devuelve un vector con los identificadores únicos de los *peers* a los que poder preguntar por dicho contenido.

- **`findContentConfidence`:**

  Este método permite realizar la búsqueda de la reputación de un contenido concreto en las tablas locales de confianza en contenidos. Para realizar esta tarea necesita que se pasen como parámetro el identificador *JXTA* del contenido, el identificador único del grupo en el que se desea hacer la búsqueda y la palabra clave a la que hace referencia el contenido.

  Devuelve como resultado un objeto de tipo `ContentConfidence` que contendrá la información solicitada. En caso de no encontrar esta información el objeto devuelto tendrá todos sus atributos vacios, o cargados con valores no permitidos para los mismos.

- **`findPeerConfidence`:**

  Permite encontrar un valor de reputación en un *peer* concreto realizando una búsqueda en la tabla local correspondiente. Para este cometido se necesita conocer el identificador único del *peer* en el que se está interesado, el identificador del grupo en el que queremos realizar la consulta, y la palabra clave sobre la que queremos conocer la reputación del *peer* remoto.

  Como resultado, este método devuelve un objeto de la clase `PeerConfidence`, en el que estará recogida toda la información que se está buscando. En caso de que la búsqueda no sea satisfactoria, el objeto devuelto contendrá valores no permitidos en sus atributos, lo que indicará que la búsqueda ha sido fallida.

- **`findPeerConfidenceGI`:**

  Este método es similar al anterior, pero con la diferencia de que la búsqueda se realiza en la tabla de confianza en *peers* independiente de grupo. Necesita como parámetros el identificador *JXTA* del *peer* que se quiere consultar y la palabra clave sobre la que se desea realizar la consulta.

  Como resultado, este método devuelve un objeto de la clase `PeerConfidence` con toda la información solicitada, o un objeto con los atributos cargados con valores no permitidos en caso de que no se haya encontrado nada en las tablas.

  Este método se suele invocar para realizar los cálculos del grado de cooperación de varios *peers*, para seleccionar a cuál de ellos solicitar la información que se necesita.

- **`findPeerConfidenceTable`:**

  Cuando un usuario es nuevo en la red, y pasa a formar parte de un grupo, sus tablas de confianza, tanto en contenidos como en *peers* estarán vacías, y por lo tanto le será mucho más difícil poder buscar información en la red, a no ser que se limite a realizar consultas sobre sus contenidos locales. En este caso, se le ofrece la posibilidad de que un *peer* remoto le facilite su propia tabla de confianza en *peers* para un grupo concreto, de forma que reciba la información de otros muchos *peers* con los que trabajar. Este es el motivo de la implementación del método `findPeerConfidenceTable`, el *peer* local puede recibir la petición de otro usuario de la red para que se le facilite el contenido de las tablas locales de confianza en *peers*, esto le dará al usuario remoto nuestra visión de la red, pero poco a poco se encargará de actualizar las tablas con sus propias observaciones.

  Este método recibe como parámetro el identificador único del grupo sobre el que se quiere conocer la tabla de confianzas y devuelve como resultado un vector con todos los objetos de tipo `PeerConfidence` contenidos en la misma, para todas las palabras clave que se conozcan. Este vector se hará llegar al *peer* remoto a través de ciertos métodos y clases que veremos más adelante.

- **`updatePeerConfidenceTable`:**

  Este método es complementario al anterior, encargándose de actualizar la tabla local de confianza en *peers*. Recibe como parámetro el identificador y el nombre del grupo cuya tabla queremos actualizar y un vector de cadenas de caracteres que contendrá toda la información de la tabla, que se tendrá que transformar en un nuevo conjunto de `PeerConfidence`.

- **`removeContentFromGroup`:**

  Este método permite borrar la reputación de un contenido que haga referencia a una palabra clave concreta. La motivación de este método se basa en que un usuario puede darse cuenta, tras acceder al contenido, que el mismo no hace referencia a la palabra clave sobre la que se realizó la búsqueda, por lo que no tiene sentido que siga en las tablas de confianza bajo esa palabra clave, lo que haría, desde el punto de vista del *peer* local, que este difundiese información incorrecta por la red.

  Este método recibe como parámetro el identificador del grupo del que se quiere borrar dicho valor de reputación, la palabra clave que consideramos errónea, y el identificador del contenido. Como resultado, el método devolverá un valor *booleano* que indicará si el contenido se ha quitado con éxito de la tabla.

- **`calcCoopThreshold`:**

  Este método se invoca cada vez que un usuario desea saber a qué *peer* remoto solicitar una información de reputación determinada, permitiendo obtener un valor numérico que indicará lo dispuesto a colaborar que estará dicho *peer*. Necesitará como parámetros el identificador del *peer* cuya cooperación se quiere solicitar y la palabra clave sobre la que se le pedirá un contenido, esto nos da una idea de que un *peer* puede mostrarse más dispuesto a colaborar si se le pregunta por un determinado tema que por otros.

  Como hemos dicho, el método devolverá un valor numérico decimal que nos indicará el grado de disposición a cooperar que tiene dicho *peer*.

- **`updatePeerRisk`:**

  Como hemos visto anteriormente, cada vez que un usuario accede a un contenido remoto y lo evalúa, se tiene que realizar la actualización de todas las tablas de confianza: la de contenidos, la de *peers*, la de *peers* independientemente de grupos y la de riesgos. Este método se encarga de actualizar la tabla de riesgos en base a la última actuación de un *peer* remoto. Recibe como parámetro el identificador del *peer* del que se desea actualizar el factor de riesgo, y la accesibilidad y rendimiento que mostró dicho *peer* en su último acceso como números decimales en el rango [-1,4].

  `updatePeerRisk` devuelve un *booleano* que indicará si la actualización del factor riesgo para dicho *peer* se ha realizado correctamente.

- **`getContentName`:**

  Este método se encarga de buscar entre las tablas de confianza en contenidos el nombre de un contenido determinado, pasándole como parámetro su identificador *JXTA* único.

- **`getContentID`:**

  Permite conocer el identificador *JXTA* de un contenido pasándole como parámetro su nombre como una cadena de texto. La búsqueda se realiza en la tabla local de confianza en contenidos.

- **`getPeerName`:**

  Devuelve el nombre de un *peer* determinado realizando una búsqueda en las tablas locales de confianza en *peers*. Para ello necesita recibir como parámetro el identificador *JXTA* único de dicho *peer*.

- **`getPeerID`:**

  Permite buscar el identificador único de un *peer* pasándole como parámetro el nombre del mismo como una cadena de caracteres. En este caso la búsqueda se realiza también en la tabla local de confianza en *peers*.

- **`getGroupName`:**

  Este método se encarga de devolver el nombre de un grupo determinado a partir del identificador único del mismo.

- **`getGroupID`:**

  Devuelve el identificador *JXTA* único de un grupo pasándole como parámetro una cadena de caracteres que representa su nombre.

- **`addContentPair`:**

  Este método añade a las tablas auxiliares de las tablas de confianza una nueva entrada nombre-identificador de un contenido.

- **`addPeerPair`:**

  Este método se encarga de añadir a las tablas auxiliares de las tablas de confianza una nueva entrada nombre-identificador de un *peer*.

- **`addPeerGroupPair`:**

  Este método añade a las tablas auxiliares de las tablas de confianza locales una nueva entrada nombre-identificador de un grupo.

- **`save`:**

  Este método es de vital importancia para la ejecución del sistema, ya que es el encargado de forzar el guardado de todas las tablas de confianza en disco. Como se vió en cada una de las tablas, este método se invocará periódicamente para evitar pérdidas de información, y al finalizar la sesión del sistema para guardar el estado último de las tablas.

- **`stop`:**

  Este método se invoca únicamente al finalizar la ejecución del sistema y su cometido es, al igual que el anterior, realizar el guardado de las tablas en disco.

D.2.9. Autosave

D.2.9.1. Atributos

- **`tableMgr`:**

  Es el mismo gestor de tablas de confianza que el empleado en la clase `Peer`. La clase `Autosave` necesita una instancia del mismo para poder acceder a los métodos de guardado de las tablas, por lo que se hace necesario tenerlo como atributo de la clase.

- **`delay`:**

  Es un número entero que expresa, en milisegundos, el tiempo que ha de transcurrir entre dos guardados consecutivos de las tablas. Hemos visto que se

configura en el arranque del sistema, pero puede ser modificado por el usuario en cualquier momento.

- **`stop`:**

  Es un atributo *booleano* que indica si el hilo lanzado por esta clase debe seguir ejecutándose (verdadero) o si el sistema debe parar (false).

D.2.9.2. Métodos

- **`Autosave`**:

  Constructor de la clase que recibe como parámetro los dos atributos principales de la clase, es decir, un objeto de tipo `TableManager` y un número entero que representa el tiempo entre guardados.

- **`setDelay`:**

  Método que se encarga de actualizar el valor del atributo `delay`. Este método se invoca cuando el usuario desea cambiar el tiempo entre guardados a lo largo de la ejecución del sistema.

- **`run`:**

  Método heredado de la clase `Thread` (véase el *API* de *Java*) que contendrá toda la funcionalidad del hilo. En este caso, el funcionamiento se basa en un bucle que, en cada ejecución, invoca al método de guardado de tablas del atributo `tableMgr`, y posteriormente duerme el hilo durante un intervalo de tiempo igual al valor del atributo `delay`.

- **`stopThread`:**

  Método que detiene la ejecución del hilo. Se invoca cuando el usuario sale del sistema y su cometido es destruir el hilo.

D.3. Módulo de gestión de logs

D.3.1. PLogger

D.3.1.1. Atributos

- **`logger`:**

  Es un objeto de la clase `Logger` proporcionada por el *API* de *Java* y, tal y como se puede observar en dicho *API*, es una clase cuyo cometido es registrar y almacenar mensajes generados por una aplicación o sistema. En este caso será la base de la clase `PLogger` y será el encargado de la gestión de dichos mensajes en nuestro sistema de reputación.

- **`fh`:**

  A pesar de que su nombre no sea demasiado descriptivo, `fh` es un objeto de la case `FileHandler`, que también podemos encontrar en el *API* de *Java*, y

cuyo cometido es el de realizar de intermediario para escribir en ficheros de disco toda la información que proporcione `logger`. Este objeto también nos permite configurar dichos ficheros de *log* aplicándoles un formato de documento *XML* que, no olvidemos, será un elemento muy importante dentro de nuestro sistema.

- **`fileName`:**

  Es una cadena de caracteres que representa el nombre del fichero *XML*. Este nombre será una combinación de la fecha y la hora en la que se creó dicho fichero, para poder saber a simple vista qué ficheros se han creado más recientemente. Cada vez que el sistema arranque se creará un nuevo fichero de *log* con un nombre distinto, eso sí, todos tendrán extensión *xml*.

- **`logDir`:**

  Es un objeto de la clase `File`, también proporcionada por el *API* de *java*, que representa la ruta completa al directorio donde se almacenarán todos los ficheros de *log*. Será un directorio situado dentro del directorio de trabajo del usuario.

- **`userPath`:**

  Será una cadena de caracteres que representa la ruta completa al directorio de trabajo del usuario.

- **`lm`:**

  De nuevo nos encontramos con un atributo cuya naturaleza no queda clara con su nombre. En este caso, `lm` es un objeto de la clase *Java* `LogManager` encargado de gestionar la coexistencia de distintos sistemas de *log* que puedan estar ejecutándose en el dispositivo en el que se ejecute el sistema de reputación. Así, lo único que se tendrá que hacer es añadir nuestro atributo `logger` a `lm`, para que este se encargue de su gestión.

D.3.1.2. Métodos

- **`PLogger`:**

  Es el constructor de la clase, que recibe como parámetro, al igual que todas las tablas de confianza, una cadena de caracteres que representa la ruta al directorio de trabajo del usuario. Este constructor se encargará de inicializar todos los atributos de la clase con sus valores correspondientes con el fin de que el objeto quede preparado para recibir mensajes de *log*.

- **`checkDir`:**

  Método auxiliar que comprueba si el directorio del *logs* ha sido creado previamente fruto de ejecuciones previas del sistema, o si ha de crearse en el momento para albergar los nuevos ficheros.

- **`getLogFileName`:**

  Este es un método auxiliar cuya importancia es vital para el sistema, ya que se encarga de obtener el nombre completo del fichero de *log* donde se almacenarán los nuevos registros. El nombre del fichero será una combinación de la fecha y la hora a la que se creó el mismo y tendrá extensión *xml*.

- **`deleteOldFiles`:**

  Se encarga de borrar los ficheros de *log* correspondientes a antiguas ejecuciones del sistema con el fin de ahorrar espacio en disco. Este método podrá ser invocado en cualquier momento por el usuario, manteniéndose únicamente el fichero en el que se está escribiendo actualmente.

- **`showLog`:**

  Este es el encargado de devolver una cadena de caracteres con toda la información recogida en el fichero de *log* hasta el momento. Recordemos que, a pesar de ser una cadena de caracteres, el resultado seguirá siendo la representación de un documento *XML*.

- **`log`:**

  Método principal de la clase, y por lo tanto el más importante. Se invoca cada vez que el sistema realiza algún tipo de operación, ya sean búsquedas locales, remotas o peticiones realizadas por otros *peers* de la red, creando un registro completo de todo lo que acontece. Recibe como parámetro una cadena de caracteres que se corresponde con una entrada del fichero de *log*. Contendrá toda la información relevante de dicha operación.

- **`setUserPath`:**

  Como hemos visto hasta ahora, el usuario puede cambiar en cualquier momento la ubicación de su directorio de trabajo. Este método hace efectivo tal cambio para empezar a trabajar en la nueva ubicación.

- **`stopLogger`:**

  Se encarga de cerrar correctamente todos los atributos de la clase, con el fin de que no se presenten inconsistencias o errores en los ficheros de *log*.

D.4. Módulo de gestión de la comunicación

D.4.1. PMessage

D.4.1.1. Atributos

- **`RRQST`:**

  Es un número entero constante, de valor 1, que define que el tipo de mensaje con el que estamos tratando es una petición.

- **RRPLY:**

  Es un número entero constante, de valor 2, que indica que el tipo de mensaje que estamos tratando es una respuesta.

- **RALRT:**

  Es un número entero constante, de valor 3, que indica que el tipo de mensaje es de alerta. Este tipo de mensajes en concreto no se va a utilizar, ya que con los dos anteriores se puede gestionar todos los tipos de mensaje necesarios. Aún así se deja codificado para futuras ampliaciones por si fuese necesario.

- **KEYW:**

  Es un número entero constante, de valor 1, que indica que el subtipo de mensaje, ya sea de tipo petición o respuesta, se refiere a la búsqueda de contenidos en base a una palabra clave.

- **CONT:**

  Es un número entero constante, de valor 2, que indica que el subtipo de mensaje, ya sea petición o respuesta, se refiere al intercambio de un valor de reputación de contenido.

- **RATE:**

  Es un número entero constante, de valor 3, que indica que el subtipo de mensaje, ya sea petición o respuesta, se refiere al intercambio de la puntuación de un contenido evaluado previamente.

- **TABL:**

  Es un número entero constante, de valor 4, que indica que el subtipo de mensaje, ya sea petición o respuesta, se refiere al intercambio de tablas de `PeerConfidence` entre dos *peers*.

- **type:**

  Es un número entero que representa el tipo de mensaje que, como hemos visto, puede ser una petición (1-`RRQST`), una respuesta (2-`RRPLY`), o un mensaje de alerta (3-`RALRT`).

- **subtype:**

  Es un número entero que representa el subtipo de mensaje. Como hemos visto puede ser de búsqueda de palabra clave (1-`KEYW`), de contenido (2-`CONT`), de puntuación (3-`RATE`) o de intercambio de tablas (4-`TABL`).

- **senderID:**

  Es un objeto de tipo `PeerID`, contenido en el *API* de *JXTA*, que representa el identificador *JXTA* único del *peer* que generó el mensaje.

- **`recipientID`:**

  Es un objeto de tipo `PeerID`, contenido en el *API* de *JXTA*, que representa el identificador *JXTA* único del *peer* destinatario del mensaje.

- **`peerGroupID`:**

  Es un objeto de tipo `PeerGroupID` (véase el *API* de *JXTA*) que representa el identificador *JXTA* del grupo de intereses en el que tiene validez el mensaje. Esto indica el grupo en el que se desea buscar un contenido, una tabla, o realizar la evaluación de un contenido.

- **`peerGroupName`:**

  Al igual que ocurre con otras clases, necesitamos una cadena de caracteres que represente el nombre del grupo anterior para facilitar la presentación de resultados al usuario.

- **`contentID`:**

  Es el identificador del contenido al que hace referencia el mensaje. Este atributo no se utiliza en todos los tipos y subtipos de mensaje.

- **`rate`:**

  Es un número entero que representa la calificación dada por un usuario a un contenido determinado. Puede tomar cualquier valor en el rango [-1,4].

- **`keyword`:**

  Es la palabra clave a la que hace referencia el mensaje, representada como una cadena de caracteres.

- **`resultSet`:**

  Es un vector de objetos de tipo `ContentConfidence`, que representa los resultados obtenidos por una búsqueda de contenidos en base a una palabra clave.

- **`stringResultSet`:**

  Es un objeto complementario al anterior. En este caso, es el mismo vector de resultados pero procesado para que cada elemento del mismo sea una representación en formato de cadena de texto con toda la información de cada objeto `ContentConfidence`. Este objeto resulta muy útil para añadir dicha información al mensaje final y poder enviarlo por la red.

- **`tableResultSet`:**

  El objetivo de este método es similar al anterior, pero en este caso se busca una representación en formato de cadena de texto, de todos los resultados obtenidos en la búsqueda de una tabla de confianza en *peers*. Cada elemento

de este vector se corresponderá con las diferentes palabras clave a las que hacen referencia los resultados.

- **`relevance`:**

  Es un número decimal que representa el nuevo valor de reputación que tiene un contenido que acaba de ser evaluado.

- **`popularity`:**

  Es un número entero que representa el grado de popularidad que tiene un determinado contenido.

- **`msgID`:**

  Es un identificador *JXTA* único del mensaje. Servirá para identificar respuestas que pertenezcan a sus correspondientes peticiones.

- **`ttl`:**

  Es un número entero que representa el tiempo de vida del mensaje, esto es, el número de saltos que dará antes de ser descartado. Su valor por defecto será 1, aunque podrá ser modificado.

- **`finalMsg`:**

  Es un objeto de tipo `Message`, que será la representación del mensaje dentro de la red *JXTA*, con todos los campos necesarios para su envío.

D.4.1.2. Métodos

- **`PMessage`:**

  Es el constructor por defecto de la clase. Este método se invocará cuando sea el propio usuario el que decida comenzar la construcción de un mensaje, ya sea para realizar una nueva petición o para responder a alguna consulta de un *peer* remoto. El método inicializará con valores por defecto todos los atributos de la clase y, posteriormente, será el sistema el encargado de asignar los valores correctos a cada uno de ellos antes de que el mensaje pueda ser enviado.

- **`PMessage`:**

  Este es el segundo de los constructores de la clase, siendo de igual o mayor importancia que el primero si cabe. En este caso se invocará siempre que el sistema reciba un nuevo mensaje de la red, y se encargará de extraer todos los campos del mensaje para que puedan ser procesados posteriormente por el sistema de gestión de reputación.

  Así, será el propio constructor el que determine de qué tipo y subtipo de mensaje se trata, la identidad del *peer* que lo envió y el resto de parámetros necesarios para que pueda ser procesado, en función de la clase de mensaje que se esté manejando en cada momento.

- **`getMessage`:**

  A pesar de ser uno de los métodos `get` de la clase, tienen una importancia mayor de la que pudiera parecer, por lo que se debe parar para explicarlo. Este método se encarga de, al contrario del constructor que acabamos de ver, condensar todos los campos del mensaje que se quiera formar, en un mensaje que podrá ser enviado por la red.

  De esta forma, `getMessage` se encargará de recorrer todos y cada uno de los atributos de la clase para componer alguno de los tipos de mensaje bien formados del sistema de reputación, que veremos más adelante.

- **`toString`:**

  Es un método de depuración que devuelve una cadena de caracteres con todos los atributos de la clase, tanto los utilizados como los que no, con el fin de comprobar que todos ellos están cargados con los valores correctos.

- **`getLogString`:**

  Es un método que se encarga de devolver una cadena de texto que se almacenará en el fichero de *log* del sistema, ya que cada mensaje recibido implica una nueva interacción con otro *peer*, y por lo tanto es susceptible de ser registrada.

- **`parseStringRSE`:**

  Cuando se recibe el mensaje correspondiente a una respuesta de tipo *KEYW*, esto es, cuando se reciben resultados de la búsqueda de contenidos remotos según una determinada palabra clave, dicha respuesta contendrá un conjunto de resultados para la mencionada búsqueda. Estos resultados vendrán dados en forma de cadena de caracteres con la siguiente estructura *"identificador de contenido;nombre de contenido"*, por lo que se necesitará realizar algún tipo de operación sobre estos resultados con el fin de obtener únicamente los identificadores de dichos contenidos. El método `parseStringRSE` se encarga de realizar esta operación y cargar el atributo `resultSet` con los identificadores encontrados.

- **`addElementToResultSet`:**

  Este método se invocará cuando se esté procesando el atributo `stringResultSet` mediante el método descrito anteriormente. Cada vez que se encuentre una nueva entrada correspondiente a un contenido encontrado, se añadirá al atributo `resultSet` una nueva entrada en forma de identificador de contenido.

- **`addElementToStringResultSet`:**

  Cuando se recibe un mensaje de tipo respuesta y subtipo *KEYW*, es decir, cuando se obtienen los resultados de una búsqueda remota de contenidos en

base a una palabra clave, dicho mensaje contendrá varios resultados que deberán ir almacenándose en el atributo `stringResultSet` de la clase. Este método se encarga de ir añadiendo todos y cada uno de los resultados a dicho atributo.

D.5. Módulo de gestión de la reputación

D.5.1. Peer

D.5.1.1. Atributos

- **`nmg`:**

  Es el gestor de la red *JXTA* propiamente dicho. De este atributo se obtendrán todos los servicios necesarios para la conexión con la red *JXTA* para el descubrimiento de nuevos usuarios presentes en la red, e incluso nos facilitará los mecanismos necesarios para el envío y recepción de mensajes a los mismos. Es una de las piezas fundamentales de la clase, ya que sin ella el *peer* estaría aislado.

- **`pg`:**

  Hasta ahora hemos tratado a los grupos del sistema de reputación como un simple conjunto de *peers* interesados por los mismos temas. Es el momento de dar un paso más y presentar el concepto de grupo dentro de la red *JXTA*. Para este sistema de reputación se hará una identificación directa entre una y otra definición de grupo. Por otra parte, en el caso del grupo desde el punto de vista de la red *JXTA*, este será un nuevo objeto que nos proporcionará todos los mecanismos necesarios para la comunicación entre los usuarios de un mismo grupo.

  Al inicio de la ejecución del sistema todos los usuarios pertenecerán a un grupo común por defecto, aunque luego podrán crear y unirse a otros grupos según sus intereses. De este grupo obtendremos todas las funciones básicas para la intercomunicación entre *peers*.

- **`ncfg`:**

  El gestor de la red *JXTA* `nmg` que se vió al principio será el encargado de dotar al sistema de todos los mecanismos necesarios para la conexión con la red *JXTA*, pero antes de que dicho gestor inicialice la red *JXTA* deberá tener en cuenta una configuración inicial representada por el atributo `ncfg`. Mediante las opciones que ofrecidas por este configurador se podrán establecer, desde los puertos que serán utilizados por el *peer* para conectarse a la red, hasta el nombre del usuario que se conectará a la misma.

  Una vez definidos todos los parámetros de configuración, se asignará este atributo al gestor de la red `nmg` que, una vez leídas las directrices marcadas

por `ncfg`, podrá inicializar la red *JXTA* y permitir al usuario que comience a hacer uso del sistema de reputación.

- **`ds`:**

  Como se puede consultar en las especificaciones de la plataforma *JXTA*, de cada grupo se podrá obtener un servicio de descubrimiento (*Discovery Service*) que será el que nos permitirá la localización de otros usuarios de la red. El atributo `ds`, es la implementación de dicho servicio, y nos valdremos de él para buscar contenidos y usuarios en la red mediante anuncios.

- **`es`:**

  Al igual que el atributo anterior, existe otro servicio de la red *JXTA* llamado servicio de extremos (*Endpoint Service*) que será modelado por un atributo de la clase `EndpointService`. Éste atributo dotará al sistema de los mecanismos necesarios para la identificación de los puntos de acceso a cada *peer* y por lo tanto permitirá realizar las comunicaciones entre usuarios para el intercambio de mensajes dentro de la red *JXTA* para el sistema de reputación.

- **`pID`:**

  El usuario, que se corresponde con el *peer* de la red *JXTA*, estará identificado unívocamente dentro de esta a través de un identificador. Este identificador será el mismo a lo largo de toda la ejecución del sistema para un usuario, incluso cuando se desconecte de la red y luego vuelva a conectarse. De esta forma siempre se sabrá que el usuario con un identificador determinado tendrá para el sistema local un valor de reputación concreto y, aunque cambie de ubicación, siempre se podrá localizar gracias a su identificador único.

  Para el caso de los *peers* este identificador se traduce como un objeto de la clase `PeerID`, proporcionado por el *API* de *JXTA*, y que no variará a lo largo de la ejecución del sistema.

- **`gID`:**

  Como se mencionó antes, todo grupo dentro de la red *JXTA* posee un identificador único que lo diferencia del resto, incluso si dos grupos tienen el mismo nombre (hecho que no debería ocurrir) siempre se podrán diferenciar por su identificador. Este identificador se toma en nuestro sistema como un objeto de la clase `PeerGroupID` (ver *API* de *JXTA*) y tendrá como valor inicial el grupo por defecto al que se conectan todos los *peers* cuando crean una instancia de la red *JXTA*.

- **`name`:**

  Todo usuario de la red podrá ser reconocido gracias a su identificador *JXTA*, pero este no puede considerarse un método demasiado amigable de cara a la presentación de información, ya que viene representado por una cadena de

caracteres y números bastante extensa. Esto puede desembocar en confusiones a la hora de identificar usuarios. Por este motivo, al igual que ocurre con los grupos o los contenidos, los *peers* también deberán acompañar su identificador único con un nombre corto o alias que lo identifique de cara al resto de usuarios pero que no sea demasiado complicado. Con este fin se define el atributo `name` como una cadena de caracteres que representa el nombre del *peer*.

- **`peerAdv`:**

  La red *JXTA* se basa, además de en el intercambio de mensajes entre *peers*, en la publicación de una serie de anuncios en la red. Cada anuncio puede representar un grupo, un servicio o un *peer*. En este caso, `peerAdv` será la representación del usuario actual del sistema como un anuncio de la red *JXTA*. Este anuncio tendrá una estructura *XML* en la que se podrá encontrar entre otros muchos datos, el nombre y el identificador único del usuario.

- **`tcpPort`:**

  La red *JXTA* tiene la ventaja de ser independiente del protocolo de transporte que utilice la red, por este motivo elegiremos el protocolo *TCP*, al que deberemos asignar un cierto puerto, representado como un número entero, en el que escuchará el usuario del sistema y a través del cual se comunicará.

- **`httpPort`:**

  Al igual que ocurre con el protocolo de transporte, *JXTA* también se puede comunicar utilizando el protocolo *http*. En nuestro caso esta opción estará desactivada pero se implementa para futuros cambios en el sistema. En este caso también es un número entero, y podría ser el mismo que el puerto *TCP*.

- **`home`:**

  Es el directorio de trabajo del usuario, representado como un objeto de la clase `File` proporcionado por el *API* de *Java*. Será el directorio en el que se almacene toda la información del usuario, desde la configuración de la red *JXTA* hasta las tablas de confianza, pasando por los ficheros de *log*.

- **`tableMgr`:**

  Se trata del gestor de tablas de confianza del sistema. Ya se ha hablado de él anteriormente, por lo que no necesita demasiada explicación.

- **`availablePeerGroups`:**

  Es un vector que recoge todos los grupos disponibles a los que pertenece el *peer*. Se trata de un conjunto de objetos de la clase `PeerGroup`, del que podremos obtener los servicios necesarios para conectarnos a diferentes *peers* y grupos en cada momento.

- **`availableGroupList`:**

  Es el mismo vector que el anterior atributo, solo que en este caso, en lugar de ser un objeto `PeerGroup` serán simplemente el conjunto de los identificadores *JXTA* de dichos grupos.

- **`currentQuery`:**

  Cada vez que el usuario desea realizar la búsqueda de un contenido en la red, debe almacenar el patrón de la petición que realizó, para comprobar posteriormente que las respuestas obtenidas de los *peers* remotos se ajustan a dicho perfil, descartando si fuese necesario aquellas respuestas que no coincidan con nuestros criterios. Este será un objeto del tipo `ContentConfidence` que contendrá la plantilla del contenido buscado, ya sea palabra clave, grupo o incluso identificador de contenido.

- **`currentQueryID`:**

  Para garantizar la seguridad en el intercambio de mensajes, cada vez que se realiza una búsqueda, se asigna a dicho mensaje un identificador único, que será el que se comprobará posteriormente para garantizar que las respuestas obtenidas se corresponden efectivamente con dicha petición. Como se ha visto, en la red *JXTA* un contenido puede representar multitud de objetos, por lo que se ha decidido que el identificador de la petición sea de tipo `ContentID`.

- **`netResultSet`:**

  Es una tabla que contiene el conjunto de todos los resultados obtenidos de la red para una petición de contenido determinada. Estará implementado como un objeto de tipo `Hashtable` ordenado en función del identificador del *peer* que facilitó la respuesta. Cada entrada de esta tabla será un vector con los identificadores de los contenidos que ofrece dicho *peer* en respuesta a la petición formulada. Observemos que este atributo tendrá sentido cada vez que se realice una búsqueda de contenidos en base a una palabra clave para acceder a los mismos.

- **`netResultTable`:**

  Es una tabla con los resultados obtenidos como respuesta a una petición de contenidos, organizada por grupos, ya que no siempre se van a encontrar resultados en el grupo deseado, y se deberá hacer uso de la información de otros grupos disponibles.

- **`propagatedContentConfTable`:**

  Cada vez que se desea evaluar un contenido y actualizar su anterior valor de reputación se deberá realizar una petición a todos los *peers* conocidos para que envíen sus opiniones acerca del mismo contenido, en el mismo grupo y

para la misma palabra clave, de forma que se recupere una idea general de la reputación de dicho contenido.

La respuesta a esta petición se agrupa en este atributo, que será una tabla ordenada por identificador del *peer* que envía la información y cuyas entradas serán objetos de tipo `ContentConfidence` con la opinión de dichos *peers*.

- **`pLogger`:**

  Es el gestor de *logs* del sistema. Es un objeto de la clase `PLogger`, ya descrito anteriormente, por lo que no es necesario parar a explicar su utilidad.

- **`waitTime`:**

  Es uno de los parámetros de configuración, o propiedades, del sistema de reputación. Estará codificado en un fichero *XML* que se leerá al inicio de la ejecución del sistema, pero podrá ser modificado en cualquier momento por el usuario.

  Es un número entero que indica, en milisegundos, el tiempo que deberá esperar un *peer* desde que realiza una petición hasta que comienza a procesar las respuestas recibidas. Este tiempo podrá ser mayor si se desea aumentar la flexibilidad y contar con un mayor número de respuestas, en caso de que haya *peers* más lentos que otros.

- **`saveTime`:**

  Es uno de los parámetros de configuración, o propiedades, del sistema de reputación. Estará codificado en un fichero *XML* que se leerá al inicio de la ejecución del sistema, pero podrá ser modificado en cualquier momento por el usuario.

  Es un número entero que indicará, en milisegundos, el intervalo de tiempo entre guardados consecutivos de todas las tablas de confianza manejadas por el sistema.

- **`ttl`:**

  Es uno de los parámetros de configuración, o propiedades, del sistema de reputación. Estará codificado en un fichero *XML* que se leerá al inicio de la ejecución del sistema, pero podrá ser modificado en cualquier momento por el usuario.

  Es un número entero que indica el tiempo de vida de los mensajes que sean generados por el sistema, es decir, el número máximo de saltos que pude dar dicho mensaje antes de ser descartado por el *peer* que lo reciba.

- **`pBest`:**

  Es uno de los parámetros de configuración, o propiedades, del sistema de reputación. Estará codificado en un fichero *XML* que se leerá al inicio de la

ejecución del sistema, pero podrá ser modificado en cualquier momento por el usuario.

Cada vez que se realiza la búsqueda de un contenido por lo general se reciben respuestas de múltiples *peers*. Para decidir a qué *peer* se le encargará el trabajo finalmente se debe hacer una ordenación de los mismos en base al grado de cooperación que muestren. Una vez ordenados de este modo el sistema optará con una probabilidad igual a `pBest` por el *peer* mejor posicionado en la tabla.

Esto puede parecer extraño ya que, en teoría siempre se debería querer acceder al *peer* con mejor comportamiento. Si fuese así, los *peers* mejor valorados acabarían por sobrecargarse, por lo que este mecanismo nos garantiza en cierto modo una distribución más o menos uniforme de la carga de peticiones que reciben los *peers* mejor valorados.

- **`userPath`:**

  Como en las clases precedentes, es una cadena de caracteres que representa la ruta completa al directorio de trabajo del usuario.

- **`autosave`:**

  Es un objeto de la clase interna de `Peer` denominada con el nombre de `Autosave`. Llevará el peso de realizar los guardados de las tablas de confianza, pero se explicará más adelante.

- **`initTS`:**

  Es una `Hashtable` en la que se almacenan los tiempos, expresados en milisegundos, en los que se enviaron peticiones a los *peers* contenidos en ella. Formará parte del cálculo del rendimiento de dichos *peers*.

- **`endTS`:**

  Es una `Hashtable` en la que se almacenan los tiempos, expresados en milisegundos, en los que se recibieron las respuestas de los *peers* contenidos en la tabla `initTS`. Mediante una simple resta se podrá obtener el nivel del rendimiento de dichos *peers*, pudiendo actualizar así su factor de riesgo.

D.5.1.2. Métodos

- **`Peer`:**

  Es el constructor de la clase. Se encarga de inicializar todos los atributos de la misma, desde el gestor de tablas hasta el de *logs*, pasando por el arranque de la red *JXTA*, sin olvidar la lectura del fichero de propiedades para configurar los valores contenidos en el mismo. Para ello lo único que se necesitará será el nombre del *peer* pasado como una cadena de caracteres.

- **`stringToIDGroup`:**

  Se encarga de convertir una cadena de caracteres que representa el identificador *JXTA* único de un grupo en un objeto de la clase `PeerGroupID`, es decir, un identificador *JXTA* propiamente dicho.

- **`getAvailableContents`:**

  Permite al usuario obtener un listado de todos los contenidos pertenecientes a un grupo que hagan referencia a una palabra clave determinada. Recibe como parámetros el identificador *JXTA* del grupo en el que se quiere realizar la búsqueda y la palabra clave a la que hace referencia, en formato de cadena de caracteres. Devuelve como resultado un vector con todos los identificadores de los contenidos que se ajustan a los parámetros de búsqueda.

- **`sendTableRequest`:**

  Envía a través de la red *JXTA* mensajes de tipo *TABL* a distintos *peers* remotos para actualizar la tabla local de confianza en *peers* de un grupo determinado. Recibe como parámetro una tabla organizada con el identificador del grupo cuya tabla se quiere actualizar, y los identificadores únicos de los *peers* a los que enviar dicha petición. Devolverá un *booleano* que indicará si se ha podido enviar la petición correctamente.

  Este método se invocará cuando un *peer* acceda a la red pero no conozca la existencia de otros en la misma, o bien cuando quiera actualizar la tabla local de confianza en *peers* antigua.

- **`removeContentFromGroup`:**

  Método heredado del interfaz `ContentIface`.

- **`addNewContent`:**

  Método heredado del interfaz `ContentIface`.

- **`addKeywordToContent`:**

  Método heredado del interfaz `ContentIface`.

- **`addPeersToTable`:**

  Método heredado del interfaz `PeerIface`.

- **`createNewGroup`:**

  Método heredado del interfaz `GroupIface`.

- **`findNewGroup`:**

  Método heredado del interfaz `GroupIface`.

- **`sendContentRequest:`**

  Envía a través de la red *JXTA* peticiones de tipo *CONT* a un grupo de *peers* remotos con el fin de recabar valores de reputación de un contenido en concreto. Este método se invocará cada vez que el *peer* desee realizar la evaluación de un contenido y necesite la opinión de otros usuarios de la red para actualizar el valor de reputación de dicho contenido.

  Recibe como parámetro una tabla organizada por identificadores *JXTA* de grupos en la que cada entrada será un vector de identificadores *JXTA* de *peers* a los que enviar la petición. Esto quiere decir que recibe, para cada grupo, el conjunto de *peers* a los que poder preguntar. Al igual que el método anterior, este devuelve un *booleano* que indicará si se ha conseguido enviar alguna petición a cualquiera de los destinatarios.

- **`sendKeywordRequest:`**

  Envía a través de la red *JXTA* peticiones de tipo *KEYW* a un grupo de *peers* remotos con el fin de encontrar contenidos nuevos en la red. Este método se invocará cuando el usuario quiera realizar una nueva búsqueda de contenidos y no se obtenga ningún resultado de las tablas locales, obligando en ese caso a pedir información a otros *peers*; o bien cuando el *peer* quiera realizar dicha búsqueda forzando al sistema para que sea remota.

  Recibe como parámetro, al igual que el método anterior, una tabla de vectores que contienen identificadores únicos de *peers* organizados por grupos. Del mismo modo, devuelve un valor *booleano* que indicará si se ha conseguido enviar alguna petición.

- **`sendRateRequest:`**

  Envía a través de la red *JXTA* una petición de tipo *RATE* a un *peer* determinado. Este método se utilizará cuando un *peer*, después de haber accedido y evaluado un contenido que obtuvo de un nodo remoto, desea enviar un *feedback* de la reputación de dicho contenido al usuario que se lo proporcionó.

  Recibe como parámetro la palabra clave a la que hacía referencia el contenido, como cadena de caracteres, el identificador del *peer* que proporcionó dicho contenido, el identificador del grupo en el que se realizó la búsqueda, el identificador del contenido del que se desea enviar el *feedback*, un valor numérico decimal que indica la nueva reputación que tiene el contenido en las tablas locales, y por último un valor numérico entero que indica la popularidad de dicho contenido en las mismas.

  Al igual que ocurría con todos los métodos anteriores de envío de peticiones, este devolverá un valor *booleano* que indicará si la petición se realizó correctamente.

- **`findInContentTable`:**

  Busca en la tabla local de confianzas en contenido los `ContentConfidence` correspondientes a todos los contenidos que hacen referencia a una determinada palabra clave, dentro de uno o varios grupos. Este método supone el primer paso de la búsqueda de un contenido por parte del usuario, es decir, la búsqueda en las tablas locales, o la que se produce cuando se recibe una petición dese un *peer* remoto.

  Este método recibe como parámetros una cadena de caracteres que se corresponde con la palabra clave a la que hacen referencia los contenidos que se están buscando, el identificador de un grupo, y un valor *booleano* que indicará si se desea realizar la búsqueda en el grupo pasado como parámetro o si por el contrario se quiere realizar en todos menos en ese.

  Ya que la búsqueda puede realizarse en varios grupos, el resultado devuelto por este método debe ser una tabla de vectores ordenados por el identificador del grupo al que pertenecen. Cada vector contendrá un conjunto de identificadores de contenido que se ajustan a la búsqueda realizada.

- **`findInPeerTable`:**

  Busca en la tabla local de confianza en *peers* los usuarios a los que poder enviar peticiones acerca de una palabra clave dentro de un determinado grupo. Este método es el segundo paso en el mecanismo de búsqueda de contenidos y se invoca cuando el método anterior no ha arrojado resultados locales y por lo tanto se necesita ampliar la búsqueda en la red.

  Este método, al igual que el anterior, recibe como parámetros una cadena de caracteres con la palabra clave buscada, un identificador de grupo y un valor *booleano* que indicará si se quiere buscar en el grupo anterior o en otros.

  El objeto devuelto por `findInPeerTable` es un poco más complicado que el correspondiente al método anterior, ya que este realiza directamente la búsqueda, enviando las peticiones correspondientes y recuperando los resultados obtenidos. Por este motivo, dicho objeto es una tabla organizada por el identificador del grupo al que pertenecen y cuyas entradas, a su vez, serán otras tablas ordenadas por los identificadores de los *peers* remotos de los que se han recibido respuestas. Estas tablas estarán formadas por los vectores de identificadores de contenidos que tiene cada uno de dichos *peers* para poder ser enviados.

- **`findInAdvertisements`:**

  Por fin llegamos al último paso de las búsquedas de contenidos en el sistema de reputación. En el caso de que los dos métodos anteriores fallen y no se encuentre ningún resultado, el sistema de reputación no se dará por vencido e intentará encontrar algún contenido que se ajuste a la búsqueda del usuario

mediante la información contenida en los anuncios que cada *peer* publica en la red, mecanismo proporcionado por la plataforma *JXTA*.

Como se ha explicado anteriormente, en las tablas locales de confianza estará toda la información recopilada a través de experiencias personales de cada usuario, por lo que en ellas habrá referencia a *peers* que de algún modo u otro se conozcan y con los que se haya tenido alguna interacción. Por otro lado, en la red *JXTA* existirán publicados tantos anuncios como usuarios conectados, tanto los conocidos como aquellos que no existen en las tablas locales. Por este motivo, se tratará de buscar entre estos anuncios *peers* nuevos de los que se puedan obtener nuevos contenidos.

El método `findInAdvertisementes` recibe como parámetros los mismos atributos que los dos métodos anteriores, ya que, como recordaremos, los tres forman parte de una misma búsqueda. Por tanto recibirá como parámetro una cadena de caracteres que representa la palabra clave a buscar, el identificador *JXTA* de un grupo y un valor *booleano* que indicará si se quiere dirigir la búsqueda a dicho grupo o si por el contrario se desea realizar sobre el resto de grupos conocidos excluyéndolo a este.

El objeto devuelto por este método, al igual que en el caso anterior, será una tabla cuyas entradas, una por cada grupo del que se hayan obtenido resultados, serán a su vez otras tablas. Estas tablas internas contendrán vectores de identificadores de contenidos, uno por cada *peer* del que se haya obtenido un resultado positivo.

- **`findContent`:**

  Método heredado del interfaz `ContentIface`.

- **`selectPeer`:**

  Cada vez que se realiza una búsqueda de contenidos de forma remota, ya sea porque no se hayan encontrado resultados locales o porque se haya forzado que sea de esta forma, se puede obtener respuesta de varios *peers* diferentes de la red. En este caso, el sistema de reputación deberá seleccionar a cuál de ellos solicitar finalmente la información y para ello se encargará de ordenarlos en función del grado de cooperación que se espera obtener de cada uno de ellos (recordemos en este cálculo entra el factor de riesgo).

  Tras la ordenación de los *peers* de mayor a menor grado de cooperación, habrá una probabilidad igual al valor almacenado en el atributo `pBest` de elegir al *peer* que encabeza dicha lista, y una probabilidad igual a 1 menos `pBest` de elegir a uno de los otros.

  Como se puede esperar, este método devolverá el identificador *JXTA* único correspondiente al *peer* al que realizaremos la petición.

- **`rateContent`:**

  Método heredado del interfaz `ContentIface`.

- **`stopPeer`:**

  Realiza un apagado ordenado del sistema de reputación, realizando la desconexión de la red *JXTA*, invocando los métodos correspondientes para el guardado de todas las tablas locales de confianza y finalizando la ejecución del sistema.

- **`processIncommingMessage`:**

  Este método, heredado de la clase `EndpointListener` del *API* de *JXTA*, es invocado automáticamente por el servicio correspondiente (`EndpointService`) cada vez que se detecta un mensaje entrante dirigido al usuario actual del sistema.

  Este método se encargará de procesar dicho mensaje, identificando tipo (petición o respuesta) y subtipo (palabra clave, contenido, puntuación o tabla) y realizando las operaciones necesarias para su gestión: desde generar las respuestas correspondientes en caso de que se trate de peticiones, hasta cargar las estructuras de datos necesarias para la presentación de la información al usuario en caso de que se trate de una respuesta.

  Como se puede suponer, junto con los métodos correspondientes a las búsquedas de contenidos, este es uno de los métodos principales de la clase, llevando todo el peso de la gestión automática de mensajes.

- **`updatePeerRisk`:**

  Cada vez que se lleva a cabo una operación exitosa con un usuario remoto, los atributos de la clase `initTs` y `endTS,` vistos al comienzo de este apartado, contendrán la información para calcular el rendimiento de dicho *peer*, a partir de sus tiempos de respuesta.

  Por lo tanto, este método se invocará cada vez que se finalice un intercambio de información, con el objetivo de actualizar la entrada correspondiente a dicho *peer* en las tablas de riesgo del sistema de reputación.

- **`getAvailableGroups`:**

  Devuelve un vector con todos los nombres de los grupos disponibles para el usuario actual, en formato de cadenas de texto.

- **`insertAdvDescription`:**

  Este método recibe como parámetro una estructura de tipo *XML* que contendrá toda la información de las palabras clave que posee el *peer* local para cada uno de los grupos a los que pertenece. Este documento será insertado en el anuncio que el *peer* publicará en la red *JXTA* para que dicha

información pueda ser encontrada por cualquier usuario de la red, incluyendo aquellos con los que nunca se ha interactuado.

- **`getPeerConfidenceValue`:**

  Método heredado del interfaz `PeerIface` cuya funcionalidad se explicará en la sección correspondiente del apartado de Interfaces del Sistema.

- **`getContentConfidenceValue`:**

  Método heredado del interfaz `ContentIface` cuya funcionalidad se explicará en la sección correspondiente del apartado de Interfaces del Sistema.

- **`documentToString`:**

  Convierte un documento estructurado con formato *XML* en una cadena de caracteres para su visualización. Recibe como parámetro la correspondiente estructura *XML* y devuelve una cadena de caracteres con dicha información.

- **`containsKeyword`:**

  Se encarga de consultar un anuncio de un *peer* obtenido de la red, y comprobar que dicho *peer* posee información sobre una determinada palabra clave para un grupo determinado. Este método se invocará cuando las búsquedas de contenidos en las tablas locales y en los *peers* conocidos no ofrezcan ningún resultado, y por lo tanto haya que recurrir a los anuncios de red para buscar dicha información.

  Recibe como parámetro el anuncio de *peer* que se quiera comprobar, la palabra clave que se quiere buscar dentro del anuncio, y el identificador *JXTA* del grupo en el que se quiere buscar. Devuelve un valor *booleano* que indicará si se ha encontrado la información buscada en el anuncio o no.

- **`identifyGroups`:**

  Al igual que el método anterior, este método se invocará cuando se haya recurrido a los anuncios de la red para la búsqueda de un contenido.

  En este caso, el método se encarga de localizar los grupos contenidos en el anuncio que contengan información para una palabra clave determinada. Recibe como parámetros el anuncio de *peer* que se desea comprobar y la palabra clave que se está buscando.

  Devuelve un vector con los identificadores *JXTA* únicos de los grupos para los que el *peer* remoto al que pertenece el anuncio posee contenidos que hacen referencia a la palabra clave buscada.

- **`descToDocument`:**

  Este método también se encarga de realizar, en cierto modo, el procesamiento de uno de los anuncios de *peer* obtenidos de la red. En este caso, el método se

encarga de convertir la descripción de los contenidos que posee dicho *peer*, obtenida directamente desde su anuncio, en un documento XML estructurado con la misma información, pero cuyo procesamiento sea más sencillo.

- **`getKeywordFromDocument`:**

  Como todos los métodos anteriores, la necesidad de este surge del procesamiento de los anuncios de *peer* obtenidos de la red. En este caso, se encarga de convertir el documento estructurado *XML*, obtenido del anuncio de un *peer*, en una tabla organizada y más fácil de manejar por el sistema. Cada entrada de esta tabla será un vector con un conjunto de palabras clave. Estos vectores estarán ordenados según el identificador *JXTA* del grupo al que pertenecen.

- **`obtainPort`:**

  Método auxiliar del sistema que se encarga de obtener un puerto *TCP* libre para poder conectarse a la red de sistema de reputación, evitando así posibles duplicidades y colisiones.

- **`isAvailablePort`:**

  Comprueba si el puerto *TCP* correspondiente al número entero que se pasa como parámetro está ocupado por algún otro usuario del sistema, en el mismo dispositivo. Devolverá un valor *booleano* que indicará si dicho puerto está libre (verdadero) u ocupado (falso).

- **`loadAvailableGroups`:**

  Cada vez que se inicia el sistema de reputación se invoca este método, cuya función consiste en inicializar y cargar los atributos de la clase, `availablePeerGroups` y `availableGroupList`, vistos anteriormente, con la información almacenada en las tablas de confianza locales.

- **`loadProperties`:**

  Al igual que el anterior, este método se invoca al iniciarse el sistema desde el constructor de la clase. Su cometido es acceder al fichero con los parámetros de configuración del sistema, y cargar sus valores en los atributos de clase correspondientes.

  Recordemos que esta configuración contiene desde el tiempo de espera para obtener respuestas, hasta el tiempo de vida de los mensajes generados, pasando por el directorio de trabajo del usuario.

- **`saveProperty`:**

  Método heredado del interfaz `AdminIface`.

- **`listProperties`:**

  Método heredado del interfaz `AdminIface`.

- **`showLog`:**

  Método heredado del interfaz `AdminIface`.

- **`deleteOldLogFiles`:**

  Método heredado del interfaz `AdminIface`.

- **`getContentName`:**

  Este método se encarga de buscar entre las tablas de confianza en contenidos el nombre de un contenido determinado, pasándole como parámetro su identificador *JXTA* único.

- **`getContentID`:**

  Permite conocer el identificador *JXTA* de un contenido pasándole como parámetro su nombre como una cadena de texto. La búsqueda se realiza en la tabla local de confianza en contenidos.

- **`getPeerName`:**

  Devuelve el nombre de un *peer* determinado realizando una búsqueda en las tablas locales de confianza en *peers*. Para ello necesita recibir como parámetro el identificador *JXTA* único de dicho *peer*.

- **`getPeerID`:**

  Permite buscar el identificador único de un *peer* pasándole como parámetro el nombre del mismo como una cadena de caracteres. En este caso la búsqueda se realiza también en la tabla local de confianza en *peers*.

- **`getGroupName`:**

  Este método se encarga de devolver el nombre de un grupo determinado a partir del identificador único del mismo.

- **`getGroupID`:**

  Devuelve el identificador *JXTA* único de un grupo pasándole como parámetro una cadena de caracteres que representa su nombre.

- **`republishAdvertisement`:**

  A lo largo de la ejecución del sistema, la información almacenada en las tablas de confianza va actualizándose constantemente, agregándose o eliminándose contenidos y palabras clave. Esta información, como se ha mencionado, se publica en la red *JXTA* para que pueda ser accedida por todos los *peers* conectados, que realizarán peticiones a nuestro sistema en base a ellas.

Por este motivo se debe mantener actualizada la información publicada y, cada vez que realicen cambios en la misma, se deberá publicar de nuevo el anuncio. Este método se encarga precisamente de esto, actualizando el anuncio del *peer* publicado mediante el servicio *JXTA* correspondiente.

- **`flushLocalAdvertisements:`**

  Como se ha comentado, la información publicada por los *peers* va cambiando constantemente, mientras que los anuncios obtenidos por el sistema se almacenan en una *cache* local para facilitar su búsqueda. Por este motivo se debe realizar una limpieza de dicha *cache* periódicamente para descartar aquellos anuncios que contengan información desactualizada. Este será el método encargado de esa tarea.